VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA
EKONOMICKÁ FAKULTA

KATEDRA APLIKOVANÉ INFORMATIKY

Aplikace monitorující dostupnost webových stránek
Monitoring Application for Web Page Availability

Student: Matěj Haša
Vedoucí bakalářské práce: Ing. Ondřej Grunt, Ph.D.

Ostrava 2020

VŠB - Technická univerzita Ostrava
Ekonomická fakulta
Katedra aplikované informatiky

Zadání bakalářské práce

Student: **Matěj Haša**

Studijní program: B6209 Systémové inženýrství a informatika

Studijní obor: 6209R017 Informatika v ekonomice

Téma: Aplikace monitorující dostupnost webových stránek
Monitoring Application for Web Page Availability

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Úvod
2. Teoretická východiska implementace monitorující aplikace
3. Současný stav řešené problematiky
4. Návrh aplikace a její vývoj
5. Závěr

Seznam použité literatury
Seznam zkratek
Prohlášení o využití výsledků bakalářské práce
Seznam příloh
Přílohy

Seznam doporučené odborné literatury:

LUDIN, Stephen and Javier GARZA. *Learning HTTP/2: A Practical Guide for Beginners*. Sebastopol: O'Reilly Media Inc., 2017. ISBN 978-1491962442.
CLARK, Dan. *Beginning C# Object-Oriented Programming (2nd Edition)*. New York: Apress, 2013. ISBN 978-1430249351.
KRIEGEL, Alex. *Discovering SQL: A Hands-On Guide for Beginners*. Hoboken: Wiley Publishing, Inc., 2011. ISBN 978-1-118-00267-4.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Ondřej Grunt, Ph.D.**

Datum zadání: 22.11.2019

Datum odevzdání: 07.05.2020

Ing. Petr Rozehnal, Ph.D.
vedoucí katedry

doc. Ing. Lenka Kauerová, CSc.
proděkanka pro studium
na základě pověření k jednání č.j.
VSB/19/050319/9900 ze dne 24. 9. 2019

Prohlašuji, že jsem celou bakalářskou práci, včetně všech příloh, vypracoval samostatně.

V Ostravě dne 14.5.2020

Matěj Haša

Poděkování

Rád bych vyjádřil své poděkování panu Ing. Ondřeji Gruntovi, Ph.D. za jeho cenné rady, věcné připomínky a čas, který mi věnoval.

Obsah

1 Úvod

Žijeme v době, ve které hrají technologie důležitou roli. Většině podniků, ať v komerční či v neziskové sféře, napomáhá počítačová infrastruktura ke zlepšení efektivity, snadnějšímu řízení zdrojů, preciznější práci a v mnohém dalším. I přes všechna pozitiva jsou přítomny i stinné stránky a jindy funkční technické řešení může přestat prokazovat požadované vlastnosti.

Již čtvrtým rokem pracuji na projektu, který pomáhá realizovat prodej produktů přes webové e-shopy do více než padesáti zemí světa. Jsem členem mezinárodního týmu, jehož zodpovědností je vývoj a provozování frameworku pro automatizované testování. Proces implementace nových funkcionalit systému funguje tak, že se nově vzniklá část neintegruje přímo do prostředí, se kterým komunikuje koncový zákazník, ale nejprve do prostředí testovacího. Zde se vše důkladně otestuje a prověří se, že daná funkcionalita může být doručena zákazníkům, a tedy nasazena na produkční prostředí. Jelikož je do testovacího prostředí často přidáván nový kód k prověření, je běžné, že se celé prostředí stane nefunkční. Abychom předešli situaci, že si nefunkční platformy všimne někdo až ve chvíli, kdy ji potřebuje využít k testování, je třeba vyvinout mechanismus, který bude její dostupnost hlídat a okamžitě zalarmuje zodpovědný tým v případě nefunkčnosti.

V následující kapitole najdeme teoreticky shrnutou problematiku, která byla již výše krátce zmíněna, a to se zaměřením na fungování protokolu HTTP, objektový model dokumentu či souboru technologií ASP.NET společnosti Microsoft. Ve třetí kapitole je rozvedena daná problematika občasné nedostupnosti testovacího prostředí s popsáním aktuálního přístupu a řešení. V kapitole čtvrté je navržena a zdokumentována samotná implementace aplikace, která napomůže s monitorováním problematického prostředí. V závěru zhodnotíme funkčnost výsledné aplikace.

Cílem této bakalářské práce je vyvinout webovou aplikaci, jež bude kontinuálně kontrolovat funkčnost webových stránek na konkrétním prostředí, přehledně zobrazovat jejich aktuální stav a v případě nedostupnosti zašle email s informacemi o poruše týmu, který je zodpovědný za nápravu.

2 Teoretická východiska implementace monitorující aplikace

V této kapitole si objasníme vybrané teoretické oblasti, které budou sloužit jako východiska během tvorby aplikace.

2.1 HTML

HTML je zkratkou anglického Hypertext Markup Language, ve volném překladu hypertextový značkovací jazyk. Jedná se o jazyk s téměř třicetiletou historií a dodnes je jednoznačně nejrozšířenějším jazykem pro tvorbu webových stránek.

2.1.1 HTML v kontextu vývoje internetových stránek

Jon Duckett (2011) ve své knize uvádí, že webový prohlížeč interpretuje HTML kód jako stránku, kterou uživatelé ve výsledku vidí. V HTML souboru udáváme strukturu a obsah webové stránky užitím takzvaných elementů. Ke grafické úpravě stránky musíme využít jazyku CSS, jenž je podrobněji popsán v následující kapitole. Pro složitější weby, zejména pokud je vyžadována dynamická reakce stránky na uživatelovu interakci, je zapotřebí použití dalších jazyků, například v současnosti je nejčastěji využíván jazyk JavaScript. Tvrzení potvrzuje vtipný výrok Erica Elliotta (2019), autora několika odborných publikací ze světa IT technologií a experta na distribuované systémy. Elliott v roce 2019 napsal, že software snědl svět, web snědl software a JavaScript snědl web („Software ate the world, the web ate software, and JavaScript ate the web“).

2.1.2 Elementy a tagy

Internetová stránka je tvořena minimálně jedním souborem s příponou .html. Tento soubor obsahuje elementy, které popisují různé druhy obsahu a jsou vyjádřeny pomocí tagů. Jako příklad můžeme uvést element patičky webu, která je na webové stránce umístěna dole a obsahuje zpravidla kontaktní informace či odkazy. Tento element se v HTML souboru označuje tagem `<footer>`. Tagy umožňují webovému prohlížeči přečíst HTML kód a přetvořit jej do grafické podoby, kterou vidíme při návštěvě webu jako uživatelé. Tag může být párový nebo nepárový. Párový tag je znázorněn dvojicí tagů, kdy začínající tag obsahuje název (`<footer>`) a uzavírající tag obsahuje navíc zpětné lomítko (`</footer>`). Dovnitř párových tagů můžeme vložit další obsah. Nepárovými tagy bývají reprezentovány prvky stránky, které nemá smysl

plnit dalším obsahem. Takovým elementem je například dělicí čára či odřádkování. Obsah se nemusí vkládat pouze mezi párové tagy, ale někdy se vkládá také do takzvaných atributů. Atributy se vyskytují u všech HTML elementů a poskytují o nich dodatečné informace nebo udávají jejich chování. Ve většině případů se atributy vyskytují ve formě jména atributu a jeho hodnoty. Pokud bychom chtěli na stránku vložit odkaz na stránky Vysoké školy báňské, potřebujeme vytvořit element odkazu, který je reprezentován tagem `<a>`. Jedním z atributů tohoto tagu je atribut href, pomocí něhož můžeme definovat stránku, na kterou se chceme odkazovat. Výsledný HTML kód by tedy vypadal takto: `<a href="https://www.vsb.cz/cs/">Odkaz na Vysokou školu báňskou</a>`. Velmi využívané atributy jsou class a id, kterými si můžeme elementy označit a díky tomu s nimi pracovat například v CSS nebo v JavaScriptu.

Nesmíme opomenout zmínit strukturu HTML kódu. Mark Pilgrim (2014) ve své knize uvádí, že stránka HTML je sérií do sebe vnořených elementů reprezentovaných pomocí tagů, které tvoří datovou strukturu strom. To znamená, že některé elementy jsou sourozenci, jiné naopak potomci a rodiče. Stejně jako do párových tagů můžeme vkládat text, můžeme tam vkládat i další elementy. Předkem všech ostatních elementů je vždy `<html>`, který má rodiče prvek document, jenž je kořenem. Stejně důležité jsou elementy head a body. Head obsahuje metadata stránky, titulek, nastavení kódování, stylování nebo reference na další soubory. V body pak najdeme samotný obsah stránky.

2.1.3 HTML5

Jedná se o pátou verzi značkovacího jazyka HTML, který je dnes podporován drtivou většinou aktuálních verzí webových prohlížečů. V páté verzi jazyka HTML byly přidány nové elementy, upraveny některé stávající a jazyk byl zjednodušen. V dřívějších verzích nebylo možné nativně pracovat s videem nebo audiem a pro tyto účely se používaly moduly třetích stran jako například Adobe Flash nebo Silverlight. HTML5 nabízí pro práci s videem element označován tagem `<video>` a pro audio `<audio>`. Vylepšena je také práce s vektorovou grafikou, pro kterou můžeme v HTML5 použít element canvas. Nově přibyly tagy, které svým názvem ulehčují orientaci v kódu. Pro hlavičku webu můžeme použít tag `<header>`, navigační menu lze zabalit do tagu `<nav>` a podobně. I díky těmto změnám je HTML5 intuitivní a příjemně se používá i začátečním vývojářům.

2.2 CSS

Kapitola pojednává o značkovacím jazyce CSS, z anglického (Cascading Style Sheets).

2.2.1 Význam a použití

Již pouze s využitím HTML můžeme vytvořit funkční internetovou stránku. V předchozí kapitole jsme však vůbec nezmiňovali grafický aspekt, pouze jsme si popsali, že HTML vytváří strukturu stránky a definuje význam jejího obsahu. Pro stylování a práci se vzhledem musíme použít jazyka CSS. Jon Duckett (2011) ve své publikaci tvrdí, že CSS nám umožňuje specifikovat, jak se elementy mají zobrazit. Jako příklad uvádí specifikaci barvy pozadí stránky, styl písma pro jednotlivé odstavce nebo nastavení všech nadpisů první úrovně na modrou barvu. CSS3 je nejnovější verzí jazyka, která dovoluje například zakulacení rohů prvků (tlačítek, tabulek, …), tvoření animací, barevných přechodů a stínování.

Pokud bychom chtěli změnit vzhled elementu, můžeme za tímto účelem využít atributu style. Samotné nastavení stylů se deklaruje pomocí názvu vlastnosti a jeho hodnoty oddělené dvojtečkou. Pokud bychom tedy chtěli zobrazit odkaz na Vysokou školu báňskou červenou barvou, přidali bychom atribut style a vlastnost color bychom nastavili na hodnotu red takto: `<a href="https://www.vsb.cz/cs/" style="color:red;">Odkaz na vysokou školu báňskou</a>`. V případě nutnosti nastavit více vlastností je možno jednotlivé vlastnosti oddělit středníkem.

2.2.2 Možnosti stylování

V předešlé podkapitole jsme si ukázali první způsob stylování. Této možnosti, která využívá atribut style, se říká jednořádkové stylování. CSS nabízí ještě dva další přístupy.

Druhým přístupem je interní CSS, kde se elementy stylují na jednom místě, a ne uvnitř tagu jednotlivých elementů. Tímto místem je párový tag `<style>`, který je umístěn uvnitř párového tagu `<head>` v souboru HTML. Na rozdíl od jednořádkového stylování musíme uvádět i specifikátor. Specifikátorem určíme, na které elementy se bude stylování aplikovat. Zacílit můžeme podle samotného typu elementu (tlačítko, odkaz, …) nebo pomocí hodnoty atributu class či id. V případě zacílení na konkrétní typ elementu se styl aplikuje na všechny elementy tohoto typu v celém dokumentu. Samotná deklarace stylu se uvádí do složených závorek ve stejném formátu jako u

jednořádkového stylování. Implementace zahnutí rohů všech tlačítek o 10 pixelů v daném HTML souboru s použitím interního CSS je znázorněna na obr. 2.1.

```
<head>
    <style>
        button {
            border-radius: 10px;
        }
    </style>
</head>
```

Obrázek 2.1 - Ukázka interního CSS (zdroj: vlastní)

V naší práci budeme využívat poslední možnost, a to takzvané externí stylování. Jediným rozdílem od interního způsobu je umístění. Zatímco v předešlé variantě jsme vkládali stylování do elementu style, u externího přístupu umisťujeme totožný obsah do odděleného souboru s příponou .css. Tato varianta stylování bývá preferována, protože jednořádkové stylování míchá kód HTML s CSS, což může být matoucí. Interní CSS má naopak nevýhodu, pokud se webová stránka skládá z více HTML souborů. Pokud bychom chtěli mít sjednocené stylování na všech jednotlivých stránkách, museli bychom styly vkládat do každého HTML souboru a stylování by se stalo redundantní. Externí stylování odstraňuje obě nevýhody. Jak již bylo zmíněno, styly máme v samostatném souboru (nebo i více souborech). V Každém z HTML souborů se na něj odkazujeme uvnitř hlavičky pomocí elementu reprezentovaným tagem `<link>`. Pokud potřebujeme změnit styl, který je použit na více stránkách, činíme tak pouze na jednom místě, v .css souboru.

2.3 Document object model

V této kapitole se zaměříme na termín document object model, který nebývá často překládán do českého jazyka a ve světě IT se nejčastěji používá pod zkratkou DOM. Jedná se o reprezentaci především HTML nebo také XML dokumentů (XML je stejně jako HTML značkovací jazyk, je však využíván především pro uchovávání a sdílení dat). Joe Marini (2002) ve své knize uvádí, že DOM nám dovoluje přistupovat k datům v HTML struktuře a manipulovat s nimi. Této vlastnosti budeme v naší práci využívat a s použitím jazyku JavaScript přistupovat k elementům z HTML, a naopak nové elementy na stránku vkládat.

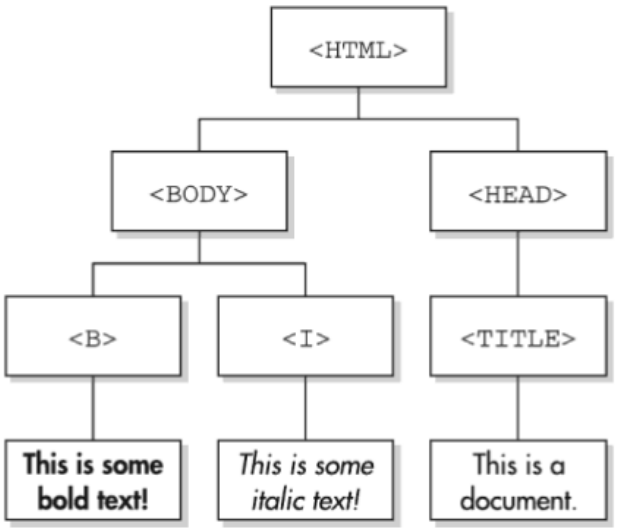
2.3.1 Document object model v kontextu HTML

Document object model nemusí znázorňovat pouze strukturu HTML, v této kapitole se ale zaměříme pouze na spojení s HTML. Jak jsme zmínili v první teoretické kapitole, HTML kód je reprezentován stromovou strukturou. Mějme jednoduchý dokument ilustrovaný na obr. 2.2.

```
<HTML>
<HEAD>
<TITLE>This is a document.</TITLE>
</HEAD>
<BODY>
<B>This is some bold text!</B>
<I>This is some italic text!</I>
</BODY>
</HTML>
```

Obrázek 2.2 - Jednoduchý HTML dokument (zdroj: Joe Marini (2002))

Joe Marini (2002) ve své publikaci a na obr. 2.3 znázorňuje korespondující stromovou strukturu k HTML dokumentu zobrazenému na obr. 2.2. Snaží se na něm ilustrovat provázanost vztahů jednotlivých elementů.



Obrázek 2.3 - DOM stromová struktura korespondující k obrázku č.2.2 (zdroj: Joe Marini (2002))

Je dobré přistupovat k HTML dokumentu a pracovat s jednotlivými elementy s vědomím toho, jaké mezi sebou mají vztahy a jak jsou provázány. Na obr. 2.3 můžeme například popsat vztah elementu BODY a elementu B jako rodič a potomek, kdy element BODY je rodičem elementu B. Vztah elementu B s elementem I slovně

popíšeme jako sourozenecký, jelikož mají stejného rodiče - prvek BODY. Těchto souvislostí budeme v praktické části využívat a shledáme je esenciálními pro náš účel.

2.3.2 Využití

Využití document object modelu může být několik. Nejenže získáme představu o návaznostech jednotlivých elementů, ale převážně je to prostředek, díky kterému můžeme manipulovat s jednotlivými elementy a modifikovat je. V naší praktické části za tímto účelem použijeme jazyk JavaScript, který bude DOM využívat k přístupu k dokumentu. Každý element je reprezentován objektem a stejně tak i celý samotný dokument. Pomocí dokumentu, který je rodičem všech elementů můžeme s využitím CSS atributů id a class (sloužící především jako identifikátory jednotlivých elementů nebo skupiny elementů) vyspecifikovat žádoucí elementy. Z nich pak lze získat potřebné informace jako například obsah vyplněného formuláře či zda je zaškrtávací box zaškrtnut a podobně.

Stejně jako můžeme získávat informace o elementech v dokumentu, můžeme jejich vlastnosti a obsah s použitím JavaScriptu a DOMu měnit. Pokud například uživatel požádá kliknutím na tlačítko o nahrání výsledků monitorující službu, můžeme k předem připravenému rodičovskému prvku připojit tolik potomků, kolik je jednotlivých výsledků. Nesmíme zapomenout zmínit skutečnost, že kromě manipulace s obsahem a s jednotlivými elementy obecně můžeme manipulovat i s jejich stylováním.

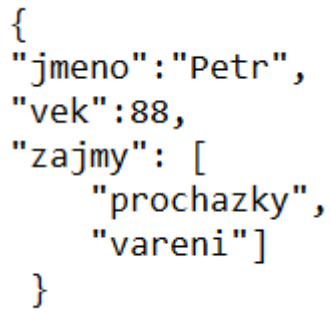
2.4 JavaScript

V této kapitole se budeme zabývat konkrétními problematikami jazyka JavaScript, který je spojován především s programováním webových stránek. V dnešní době však není výjimkou ani jeho použití na straně serveru.

2.4.1 JSON

Název JSON je zkratkou anglického JavaScript Object Notation. Jedná se o standard, který udává formát dat, a to jako strukturu párů atribut a hodnota. Tom Marrs (2017) ve své knize píše, že datový formát JSON umožňuje aplikacím komunikovat napříč počítačovými sítěmi, typicky skrze REST API. Dále udává, že všechny moderní programovací jazyky (Java, JavaScript, C#, Python …) umí s JSON formátem pracovat a podporují takzvanou serializaci (produkování dat ve formátu JSON) a deserializaci (čtení dat ve formátu JSON a případná transformace např. na objekty).

Ray Rischpater (2015) ve své knize uvádí, že data jsou organizována v párech jako atribut a hodnota atributu, v případě více párů jsou tyto odděleny čárkou. Dále zmiňuje, že jako hodnota atributu je povolen textový řetězec, číslo (i s desetinnou čárkou), pole (hodnoty odděleny čárkou v hranatých závorkách), pravdivostní hodnoty true/false a celé objekty. Objekty mohou obsahovat další pár či páry atributů a hodnot. Nesmíme zapomenout na fakt, že za JSON datový formát se považuje i například jedna hodnota textového řetězce, čísla, čísla s plovoucí řádovou čárkou, objektu, pole nebo pravdivostní hodnoty. Příklad dat ve formátu JSON je na obr. 2.4.



Obrázek 2.4 – Data ve formátu JSON (zdroj: vlastní)

2.4.2 Práce s HTML Document Object Modelem

V kapitole 2.3 jsme si vysvětlili co Document Object Model znamená a jaká je jeho struktura. V této podkapitole si ukážeme, jak se s ním pracuje za použití JavaScriptu. Ve chvíli, kdy stránka dokončí své načítání, vytvoří se DOM s prvkem document jako kořen. Právě document, jenž je předek všech elementů ve struktuře, se velmi často využívá ke hledání a cílení konkrétních elementů.

JavaScript s využitím DOM může přidávat nebo odstraňovat elementy v HTML struktuře, měnit jejich CSS stylování nebo obsah, reagovat na události HTML elementů a další. Základními nástroji jsou vlastnosti a metody. Každá vlastnost HTML elementu má hodnotu, kterou můžeme přečíst nebo změnit. Jednou ze základních vlastností je `innerHTML`, která nám umožní přistoupit k obsahu elementu. Metody naopak představují akce, které můžeme provést. Akcí může být například vyhledání konkrétního elementu, přidání elementu atp. Pokud bychom chtěli změnit obsah tlačítka, jehož atribut id je roven hodnotě “tlacitko“, provedeme tak následovně: `document.getElementById("tlacitko").innerHTML = "zmeneny obsah";`. Z celého dokumentu jsme si metodou `getElementById` získali element, jehož atribut id

má hodnotu “tlacitko“ a pomocí vlastnosti `innerHTML` jsme tlačítku změnili obsah na text “zmeneny obsah”.

S využitím metod `createElement` můžeme vytvořit nový element a uložit si ho do proměnné. Vlastnost `innerHTML` nám umožní do něj vložit obsah, a další vlastnosti přiřadit atribut id nebo class a nastylovat jej. Takto vytvořený element poté pomocí metody `append` přidáme k rodičovskému elementu. Tímto způsobem můžeme dynamicky vytvářet obsah webové stránky.

2.4.3 Fetch API

JavaScript nabízí řadu možností, jak pracovat s HTTP požadavky a zpracovávat obdržené odpovědi. Dříve se hojně využíval Ajax, alternativou může být také použití populární knihovny jQuery. My se však v této podkapitole zaměříme na Fetch API.

Fetch API navrací tzv. promisy (do češtiny možno volně přeložit jako sliby), které slouží k psaní asynchronního kódu. Jádrem Fetch API je funkce fetch, která vytváří asynchronní požadavky. Funkce má jeden povinný parametr a tím je zdroj, na který chceme požadavek poslat. Má i nepovinný parametr, ve kterém můžeme specifikovat metodu HTTP požadavku, hlavičku nebo tělo.

Již jsme zmínili, že funkce `fetch()` vrací promise. S využitím funkce `then()` můžeme počkat na navrácení výsledku asynchronní funkce `fetch()`. Jak můžeme vidět na příkladném použití znázorněném na obr. 2.5, jednotlivé funkce se dají i řetězit.

```
fetch('https://jsonplaceholder.typicode.com/todos/1')
    .then(odpoved => odpoved.json())
    .then(dataVeFormatuJson => console.log(dataVeFormatuJson))
```

Obrázek 2.5 - Příkladné využití funkcí fetch a then (zdroj: vlastní)

V tomto příkladu jsme vytvořili a zavolali GET požadavek (pokud není uvedeno jinak je GET automaticky použit jako metoda HTTP požadavku) cílený na adresu https://jsonplaceholder.typicode.com/todos/1. HTTP odpověď jsme si pojmenovali jako “odpoved“ a pomocí tzv. arrow funkce (arrow function) jsme extrahovali z odpovědi pouze tělo ve formátu JSON. Tuto hodnotu jsme následně vypsali do konzole.

2.5 HTTP protokol

Hypertext transfer protocol (HTTP) je protokol umožňující transport dat, a to převážně HTML dokumentů v relaci klient-server.

2.5.1 Protokol a vrstevnatý model obecně

Abychom byli schopni porozumět problematice protokolu HTTP, musíme si nejdříve vysvětlit, co pojem protokol znamená. Protokol můžeme chápat jako standardizovaný způsob elektronické komunikace mezi dvěma uzly počítačové sítě. Protokol může být realizován za použití hardwaru, softwarově či obojím současně. V dnešní době se přenos dat po internetu podobá referenčnímu modelu OSI (z anglického Open Systems Interconnection), vydaným mezinárodní organizací pro normalizaci (ISO - z anglického International Organization for Standardization), který se skládá ze sedmi vrstev. Vytvořením vrstevnatého modelu se komplexní problém přenosu dat v počítačových sítích rozpadne na vrstvy, kde každá vrstva řeší konkrétní část celkové problematiky přenosu. Přehled vrstev referenčního modelu OSI je znázorněn na obr. 2.6.

Obrázek 2.6 - Referenční model ISO OSI (zdroj: vlastní)

Tento model je pouze referenční, byl vydán na začátku osmdesátých let, tedy v období, kdy se již spoustu let realizovala síťová propojování počítačů. Kolektiv autorů IBM Redbooks (2001) uvádí, že sada protokolů TCP/IP, která je stavebním kamenem internetu, získala dnešní podobu již kolem roku 1978. Dále tvrdí, že první reálné implementace internetu za použití TCP/IP se datují okolo roku 1980, tedy již v době před vydáním referenční sady protokolů OSI. Rozdílem reálného použití oproti referenčnímu modelu může být například implementace aplikační, prezentační a relační vrstvy, které jsou v dnešní době realizovány zpravidla samotnou aplikací, a proto bývají ve vrstevnatých modelech TCP/IP reprezentovány pouze jednou vrstvou – aplikační.

2.5.2 Části HTTP požadavků a odpovědí

Výše jsme uvedli jeden z důvodů, proč se přenos dat v počítačových sítích dělí na jednotlivé vrstvy, a tím je dekompozice problému. Protokol HTTP řadíme do vrstvy aplikační. Douglas E. Comer (2009) uvádí, že aplikační vrstva specifikuje formát a význam zpráv, které si dvě aplikace navzájem předávají, když spolu komunikují. Může se jednat například o posílání emailů, souborů, prohlížení webových stránek, realizování videokonference a podobně.

Také jsme již zmínili, že se tento protokol řadí mezi tzv. klient-server protokoly, což znamená, že převážná většina komunikace je ve formě zaslaných požadavků od klienta na server a následných odpovědí serveru. Abychom byli schopni popsat z čeho se požadavky a odpovědi skládají, musíme zmínit ještě jeden důležitý prvek - metody HTTP protokolu. Klient může server kontaktovat z různých důvodů, mezi které patří získávání dat, změna existujících dat nebo uložení či odstranění dat. Tuto informaci klient zprostředkovává formou takzvaných metod, mezi které patří například GET, POST, PUT, HEAD, DELETE a další. Například HTTP požadavek poslaný na danou URL (z anglického Uniform Resource Locator) s metodou GET žádá server o poslání určitých dat klientovi. URL definuje zdroj a skládá se z použitého protokolu, adresy nebo jména hostitele, čísla portu, absolutní cesty a volitelných parametrů.

Požadavek i odpověď protokolu HTTP se skládají ze tří částí. Požadavek se skládá z typu, který obsahuje metodu požadavku, URL a protokolu s jeho verzí. Další částí je takzvaná hlavička, která obsahuje dvojice ve formátu klíčového slova a hodnoty. Tyto dvojice obsahují doplňující informace o požadavku jako je specifikace akceptovaných MIME (z anglického Multipurpose Internet Mail Extensions) typů na straně klienta, akceptace kódování, zaslané hodnoty cookies serveru, autorizace klienta a tak dále. Třetí část je tělo požadavku, ve kterém mohou být přenášená data. Při GET požadavku je tělo prázdné, ale například při metodě POST se v těle požadavku zpravidla nacházejí data požadovaná k uložení na straně serveru. Odpověď má podobné části, první z nich se ale liší. První část odpovědi serveru obsahuje protokol a jeho verzi, status kód a popis daného status kódu. Problematika status kódů je v naší práci důležitá, a proto je jí vyhrazena poslední část této kapitoly pojednávající o HTTP. Stejně jako u požadavku i u odpovědi je druhou částí hlavička obsahující dodatečné informace od serveru a poslední částí je tělo, které obsahuje samotná data.

2.5.3 Historie HTTP

První verzí protokolu HTTP se stala roku 1991 verze 0.9, která uměla zprostředkovat pouze metodu GET, a vůbec zde nebyl přítomen koncept hlaviček. Další verzí byla o pět let později verze 1.0. Thilina Ashen Gamage (2017) ve svém článku uvádí, že tato verze oproti předchozí disponuje možností odesílat i jiný obsah než HTML a navíc mají požadavky i odpovědi hlavičky. Také už můžeme poslat požadavek s metodou POST nebo HEAD. Nástupcem verze 1.0 se stala verze 1.1, která přinesla další rozšíření kolekce metod o PUT, DELETE, TRACE, OPTIONS a CONNECT. Umožňuje také specifikovat v hlavičce udržení spojení užitím dvojice `Connection: Keep-Alive`. Nejnovější verzí je verze HTTP/2 často zjednodušována na pouhé h2. Steven Ludin a Javier Garza (2017) ve své knize uvádějí, že od roku 1999, kdy byla uvedena verze 1.1, se internetové stránky razantně změnily. Dnešní weby obsahují bohatý obsah, a ne pouze text, jak tomu bývalo ve většině případů před dvaceti lety. Také nároky uživatelů na výkonnost se změnily a devadesátá léta, kdy lidé byli ochotni čekat na načtení stránky více než 5 vteřin jsou nenávratně pryč. Tyto důvody uvádějí Steven Ludin a Javier Garza (2017) jako spouštěč pro vytvoření HTTP/2, který je dnes již podporován řadou nejpoužívanějších webových prohlížečů a běžně ho využívají firmy jako je například Facebook, Google, Twitter nebo třeba Wikipedia.

2.5.4 Status kódy

Status kód (někdy také stavový kód) je navrácen s odpovědí od serveru klientovi. Jedná se o trojmístná čísla, která klientovi signalizují, jak byl požadavek serverem zpracován. Spolu s kódem server posílá i jeho textovou interpretaci. První číslice daného trojčíslí napovídá charakter stavu. Pokud trojice čísel začíná číslicí 1, jedná se o informační stavy. V případě, že trojice začíná číslicí 2, jedná se o úspěšné zprocesování požadavku ze strany serveru. Například status kód 200 má interpretační text “OK”. Tento kód je jedním z nejčastějších status kódů, jelikož se vrací při bezproblémovém načtení webové stránky. Trojice začínající číslicí 3, nám naznačuje přesměrování požadavku na jinou adresu. Konkrétně například kód 301 signalizuje permanentní přesměrování a URL, na kterou by se měl klient nově odkazovat zašle server v hlavičce s využitím klíčového slova Location. Trojice začínající číslicí 4 signalizují chybu na straně klienta. Z této škály můžeme vybrat například status kód 400, značící špatně zadaný požadavek, kterému server neumí porozumět. Poslední skupinou jsou trojčíslí začínající na číslici 5. Tato značí chybu na straně serveru.

Například status kód 500 má slovní interpretaci “Internal Server Error” a označuje chybový stav, který server neumí ošetřit.

2.6 C# a ASP.NET Core

C# je objektově orientovaný jazyk vyvíjený společností Microsoft. Využívá se k vývoji webových, desktopových a mobilních aplikací nebo her. Pomocí technologie Blazor dnes není výjimkou ani vývoj grafického rozhraní na straně klienta.

ASP.NET Core je framework pro implementaci webových aplikací a služeb. Stejně jako jazyk C# i tento framework je produktem společnosti Microsoft. Jedná se o poměrně novou technologii, jejíž počátek se datuje k roku 2016.

2.6.1 Properties v C#

Jelikož se jedná o objektově orientovaný jazyk, vyskytují se zde třídy. Vlastnosti tříd jsou reprezentovány proměnnými. V jazyce C# se těmto proměnným říká fieldy. Aby k proměnným neměl přístup nikdo zvenčí a zachovali jsme tak princip zapouzdření, definujeme je jako privátní. Název privátních proměnných by podle nepsané konvence měl začínat podtržítkem následovaným jménem proměnné, kde jméno nezačíná velkým písmenem. Abychom umožnili čtení a zápis privátních fieldů zvenčí, je třeba definovat metody, které tyto akce obstarají.

V programovacím jazyce C# nedefinujeme metody, ale takzvané property. Dan Clark (2013) ve své knize zmiňuje, že uvnitř bloku property se nachází blok get a set. Dále píše, že blok get vrací hodnotu privátního fieldu a uvnitř bloku set naimplementujeme logiku nastavování hodnoty privátního fieldu. Funkcionalita property zpříjemňuje celý proces přístupu k privátním proměnným, jelikož koncový uživatel nevolá metody, ale nastavuje jednotlivé property a metody se volají na pozadí. Příklad deklarace property, její nastavení a získání hodnoty můžeme vidět na obr. 2.7 a 2.8.

```
private string _jmeno;

0 references
public string Jmeno
{
    get { return _jmeno; }
    set { _jmeno = value; }
}
```

Obrázek 2.7 – Příklad deklarace property Jmeno (zdroj: vlastní)

```
var jmeno = osoba.Jmeno; // přístup k fieldu pomocí property
osoba.Jmeno = "noveJmeno"; // nastavení fieldu pomocí property
```

Obrázek 2.8 – Příklad práce s property deklarované na obr. 2.7 (zdroj: vlastní)

2.6.2 Entity Framework Core

Entity Framework Core (někdy také EF Core) je multiplatformní (tj. nelimitovaná na systém Windows) verze frameworku Entity Framework, který zprostředkovává přístup k datům a jejich transport. Umožňuje uživatelům přistupovat k databázi prostřednictvím objektů v kódu a minimalizuje tak potřebu psát SQL dotazy.

Alex Kriegel (2011) ve své knize uvádí, že SQL je programovací jazyk určen k manipulaci s daty v relačních databázích. Relační databáze nám tedy data uchovají, ale pokud je budeme chtít získat nebo měnit, je potřeba využít SQL dotazů. S využitím Entity Framework Core však nemusíme SQL dotazy psát ručně, ale můžeme si je nechat generovat.

Jedním z případů jsou takzvané migrace, které udržují databázové schéma a náš model v aplikaci synchronizován. Migrace můžeme používat jako tzv. code-first (nejdříve kód) nebo database-first (nejdříve databáze). Druhý zmíněný přístup se používá především tehdy, máme-li databázi a její schéma vytvořeno a potřebujeme tento stav zrcadlit do aplikace. V naší praktické části budeme využívat code-first přístup, kdy si v aplikaci pomocí tříd vytvoříme databázovou strukturu, kterou pomocí migrací vytvoříme v databázi. Vytvoření nové migrace je velmi jednoduché, a to napsáním `dotnet ef migrations add PrvniMigrace` do konzole. Migrace se vytvoří do složky Migrations a dalšími příkazy můžeme prostřednictvím konzole jednotlivé migrace aplikovat nebo zpětně negovat.

Zatím víme, jak udržovat konzistenci mezi třídami v aplikaci a schématem v databázi. Pro práci s daty v databázi je zásadní třídou Enity Framework Core třída DbContext. Tato třída, resp. její potomek spravuje spojení s databází a je využívána k manipulaci s daty v databázi. DbContext se využívá jako rodič třídy, kterou si vytvoříme. Ve vytvořené třídě si poté nadefinujeme property generické třídy `DbSet<T>`, reprezentující tabulky v databázi a s jejich pomocí budeme dále s těmito tabulkami pracovat. Parametr T bude poté model, námi vytvořená třída, která reprezentuje záznam v tabulce (jednotlivé sloupce, omezení pro dané sloupce atp.). Property třídy, která dědí z DbContext se využívají k interakci s jednotlivými databázovými tabulkami a Entity

Framework Core v pozadí vytváří SQL dotazy, zatímco my pracujeme v kódu na úrovni objektů.

2.6.3 ASP.NET Web API

ASP.NET Core je multiplatformní framework, díky kterému můžeme vyvíjet webové aplikace a služby. V této podkapitole se zaměříme konkrétně na ASP.NET Web API, a tedy framework sloužící k vytváření REST API služby.

Ve struktuře ASP.NET Web API projektu jsou esenciální kontrolery, třída Startup a třída Program. Kontrolerem je myšlena jakákoli třída, která dědí z třídy ControllerBase. Kontrolery reagují na požadavky klientů, a to pomocí předem nadefinovaných metod. Skutečnost, jaký kontroler se zavolá záleží na cílové adrese klientova požadavku. Relativní cestu kontroleru můžeme určit atributem třídy Route. Jednotlivé metody (znázorňující tzv. endpointy služby) označíme atributem, který symoblizuje, jaká metoda HTTP požadavku má být k jeho zavolání přítomna (pokud jsou předány i potřebné parametry metody). Příklad metody obsluhující GET požadavek s předaným číselným parametrem reprezentující id osoby lze vidět na obr. 2.9.

```
[HttpGet("{id}")]
0 references
public async Task<ActionResult<Osoba>> GetOsoba(long id)
{
    var osoba = await _context.Osoby.FindAsync(id);

    if (osoba == null)
    {
        return NotFound();
    }

    return osoba;
}
```

Obrázek 2.9 – Endpoint, který navrátí instanci osoby, jež má id totožné s předaným parametrem. Pokud taková osoba není, navrátí status kód 404 - Not Found. (zdroj: vlastní)

Další významnou třídou je třída Program, ve které se nachází statická metoda Main. Pokud spustíme webovou aplikaci, je to právě tato metoda, která se spustí jako první. Uvnitř této třídy také definujeme startovací třídu a tou bývá podle nepsané konvence třída s názvem Startup. Uvnitř třídy Startup se nacházejí dvě důležité metody. První z nich má název ConfigureServices, ve které konfigurujeme služby používané při běhu aplikace. Konkrétně se může jednat o služby běžící na pozadí, generování dokumentace pomocí často využívané knihovny Swagger atd. Druhou důležitou metodou třídy Startup je metoda Configure. V této metodě specifikujeme, jak naše

služba bude zpracovávat jednotlivé požadavky, a to pomocí přidávání middleware komponent. Uvnitř metody Configure například nastavujeme, že chceme k vyřizování požadavků využít námi nadefinovaných endpointů uvnitř kontrolerů, zda je potřeba se autentizovat a tak podobně.

2.7 DTAP

DTAP je zkratkou anglických slov Development, Testing, Acceptance a Production. Při vývoji softwaru často požadujeme vytvoření a uvedení do provozu nové funkcionality nebo opravení již existující, ale chybné části programu. Během takové modifikace kódu může nastat spousta nečekaných problémů, které ve výsledku mohou způsobit až kompletní nedostupnost celé aplikace. Z tohoto důvodu není správné přidávat daný kód okamžitě do ostrého provozu k zákazníkovi na tzv. produkční prostředí a je výhodné jej nasadit postupně přes řadu jiných prostředí. Jak už název napovídá, doporučovaná posloupnost prostředí je vývojové, testovací, akceptační a produkční.

2.7.1 Prostředí

Abychom si uměli lépe představit celý proces, musíme si blíže specifikovat, co softwarové prostředí obsahuje a jakou hraje roli. Jedná se o počítačový systém, ať už z pohledu hardwaru či softwaru, ve kterém je nasazována a provozována daná aplikace. Ta může být provozována za účelem služby pro koncové zákazníky (aplikace na produkčním prostředí) nebo například z důvodu jejího testování (aplikace na testovacím prostředí).

2.7.2 Význam různých prostředí

Goodyear (2013) tvrdí, že není žádoucí, aby vývojáři narušili chod aplikace (v našem případě e-shopu), zatímco zkouší opravit problém nebo realizují své nápady. Dále uvádí, že tohle je dobrý důvod, proč bychom neměli nechat vývojáře vyvíjet a nasazovat kód přímo na produkčním prostředí.

Z tohoto důvodu máme potřebu mít více prostředí, a ne pouze produkční, jež hostí chod aplikace, se kterou interagují koncoví zákazníci. Bylo by více než výhodné mít prostředí kde programátoři mohou vyvíjet nové části bez obav, že způsobí nefunkčnost aplikace v ostrém provozu. Tuto potřebu uspokojuje vývojové (development) prostředí, kde aplikace běží pouze za účelem vývoje. Ve chvíli, kdy je řešení hotové, může se kód nasadit na další prostředí, kde se podrobí testovací fázi –

testovací (testing) prostředí. V případě, že se během fáze, která se snaží najít přítomnost chyb, neprojeví žádná disfunkce, kód se nasadí na prostředí další, ve kterém se zpravidla kontroluje integrita s celou aplikací. Software obecně bývá velmi komplexní systém a změna v jedné části může způsobit nefunkčnost nebo neočekávané chování v místě zcela jiném. Během vývoje se může stát, že vývojové prostředí není nastaveno přesně tak, jak je nastaveno koncové produkční prostředí, kde je žádoucí bezchybné chování softwaru. Z tohoto důvodu hraje důležitou roli akceptační (acceptance) prostředí, které by mělo být nakonfigurováno co nejpřesněji tak, aby se podobalo prostředí produkčnímu a simulovalo tak jeho chování.

Přestože principy jsou podobné, počet a názvy prostředí se můžou lišit. Jak píše Goodyear (2013), počet prostředí používaných pro nasazování kódu záleží na vašich procesech a na tom, jak disciplinovaní jste při testování.

2.7.3 Vývojové prostředí

Vývojové prostředí, jak už název napovídá, využívají především vývojáři k programování nových komponent nebo k opravám již existujících. Každý vývojář má na své pracovní stanici své vlastní prostředí. Jeho architektura bývá značně zjednodušena oproti produkčnímu a obsahuje navíc vývojové nástroje. Goodyear (2013) zmiňuje, že by bylo drahé a zpomalovalo by to proces vývoje, pokud bychom udržovali přesný duplikát produkčního prostředí a jeho dat pro každé vývojové prostředí.

2.7.4 Testovací prostředí

Testování je bezesporu velmi důležitá fáze vývoje softwaru, a proto i v celém procesu publikování nové verze aplikace je vhodné ji podrobit testování. Oddělené prostředí pro testování je výhodné např. v případě, že je daná aplikace konfigurovatelná (například pomocí CMS). Aplikaci si pak můžeme pozměnit zapnutím či vypnutím libovolných funkcionalit za účelem otestování jejich integrity, aniž bychom někomu narušovali a měnili prostředí, se kterým pracuje a potřebuje ho mít nastaveno v konzistentním stavu. Vše se tedy děje odděleně na prostředí pro testování určeném.

2.7.5 Akceptační prostředí

Jelikož předešlá prostředí mohou být od produkčního odlišná, je při testování nových verzí aplikace záhodno, aby testování probíhalo na prostředí, které se pokud možno co nejvíce podobá produkčnímu. Akceptační prostředí by mělo plnit právě tento úkol, a tedy „zrcadlit“ prostředí produkční, jak konfigurací, tak i obsahem dat. V této

fázi se akceptují provedené změny a po zdárném testování putují jako nová verze aplikace ke koncovým zákazníkům.

2.7.6 Produkční prostředí

Na produkční prostředí se kód dostává ve finální fázi vývoje. Zde běží aplikace využívaná koncovým uživatelem. Na kvalitu produkčního prostředí obecně klademe velký důraz, jelikož se kvalita prostředí odráží do kvality samotné aplikace.

3 Současný stav řešené problematiky

V této kapitole se zaměříme na charakteristiku problematické části testovacího procesu, jež byla zmíněna v první kapitole ve zjednodušené podobě. Zhodnotíme nabízená řešení na trhu a vytyčíme naše cíle.

3.1 Procesy testováni softwaru v podniku

V našem podniku máme čtyři důležitá prostředí. Prvním z nich je vývojové prostředí, kde vývojáři z jednotlivých týmů implementují nové části aplikace, nebo modifikují již existující kód. Když jsou se svým výsledkem spokojeni, nasadí kód na tzv. UAT prostředí (z anglického user acceptance testing). Zde se angažují převážně manuální testeři daných týmů a podle předem nadefinovaných testovacích scénářů ověřují funkčnost aplikace.

Další v procesu je prostředí, kterému říkáme staging. V posloupnosti nasazování nových verzí se nachází před produkčním a mělo by se od něj co možná nejméně lišit. Využívá ho náš tým automatizovaného testování a také tým, který má na starosti tzv. regresní sadu. Regresní sada je sada v našem případě automatizovaných testů, které se pouštějí vždy, když je z UAT nasazena na staging nová verze aplikace. Gopalaswamy a Desikan (2008) tvrdí, že regresní testování je provedeno v případě jakýchkoliv změn v softwaru, aby se zajistilo, že tyto změny nijak nepříznivě neovlivnily již existující funkcionality. Nestačí tedy regresní testy pouze spustit, ale je nutná i jejich analýza a následné vyhodnocení výsledků testování. Tým, který je zodpovědný za regresní sadu testy spouští, když je potřeba, a zároveň zodpovídá za implementaci nových testů a údržbu starých. Velmi často se na stagingu angažuje i InfraOps tým (z anglického Infrastructure Operations), jehož úkolem je správa všech prostředí. Náš tým automatizovaného testování se stagingem pracuje, protože vyvíjí a udržuje framework pro implementaci automatizovaných testů, které se zde pouští například jako součást již zmíněné regresní sady. Je tedy často potřeba nové funkcionality nebo opravené části frameworku na stagingu otestovat.

3.2 Staging a jeho problém

Jak již bylo zmíněno v předešlé kapitole, staging je poměrně rušné prostředí, které je využíváno řadou týmů a je nezbytně nutné v celém procesu vývoje softwaru. Jelikož je žádoucí, aby co nejvěrohodněji simulovalo stav a konfiguraci produkčního

prostředí, v pravidelných intervalech se z prostředí produkčního na staging nahrávají data. V tomto kontextu si pod pojmem data můžeme představit záznamy v databázi obsahující například uživatele, objednávky, adresy, místa určena pro doručení zásilek a podobně. Reálné údaje z produkčního prostředí se anonymizují, ať už z etických či praktických důvodů (např. není žádoucí během testování odeslat email na reálnou emailovou adresu).

Proces nahrávání dat na staging spolu s nasazováním nových verzí aplikace z předešlého prostředí (UAT) zvyšuje riziko nefunkčnosti stagingu. Jak již bylo zmíněno v předešlé podkapitole, na tomto prostředí se spouští regresní sada. Konkrétně se testy spouští na dvanácti nejvytíženějších zemích (vytíženost hodnotíme dle četnosti návštěv, počtu objednávek apod.). Jelikož podnik prodává výrobky po celém světě, servery nejsou umístěny na jednom geografickém místě, ale v různých zeměpisných regionech. Například španělská verze e-shopu je umístěna na serveru, který spravuje e-shopy zemí v regionu západní Evropy. Nebylo by žádoucí z důvodu velké latence, aby španělští zákazníci pro navštívení e-shopu museli kontaktovat server fyzicky umístěn například v Latinské Americe, který naopak kontaktují zákazníci z Mexika. Ne vždy se úpravy v aplikaci nasazují na všechny servery. Může se stát, že nová verze aplikace je určena pouze pro specifický region, jelikož například marketingové kampaně, systémy nákupních bonusů atp. mohou být v každém z regionů odlišné z důvodu rozličné mentality národů a tyto systémy vyžadují rozdílnou míru a formu údržby. I když se jedná o nasazení upravené komponenty aplikace pouze pro pár regionů nebo o nasazení nové verze celé aplikace na všechny regiony, vždy je žádoucí maximální efektivita celého procesu. Konkrétně je tím myšleno co nejrychlejší spuštění regresní sady po nasazení aplikace na staging, vyhodnocení výsledků regresních testů a vyrozumění zodpovědných týmů o stavu nové verze. Bohužel se občas stává, a to především v případě nasazení nové verze aplikace na všechny regiony, že některý z regionů vykazuje nefunkčnost. Například nastane, že všechny e-shopy lokalizované na serveru v Latinské Americe jsou nedostupné a vrací HTTP status kód 503 (služba nedostupná), protože nová verze aplikace nepočítala se specifickým nastavením tohoto regionu. Může se stát, že chyba při nasazování nové verze se promítne přímo v neúspěšném nasazení, a nová verze se pro tento region vůbec nenasadí. V některých případech se může verze nasadit a nedostupnost je patrná až po samotné interakci s e-shopem, například pouhým načtením e-shopu.

Potřebujeme se vyhnout situaci, v níž se defekt zjistí v okamžiku, kdy se má již začít s testováním. Značná část času by byla ušetřena, pokud by se zautomatizovalo monitorování stavu jednotlivých zemí, na kterých se spouští regresní sada a v případě výpadku by se okamžitě kontaktoval InfraOps tým. Ten by tak byl hned informován a mohl zahájit kroky vedoucí k nápravě. Předešli bychom tak situaci, kdy se spustí regresní sada pro všechny regiony a až v průběhu analýzy výsledků testů se zjistí, že region není vůbec dostupný a je potřeba upravit verzi a znovu ji nasadit. V takovém případě se celý proces vývoje zpomaluje.

Nedostupnost e-shopu nemusí být vždy spojena s nasazováním nové verze aplikace. Nefunkčnost může zapříčinit například porucha na serveru či jiná závada na hardwaru nutném k plynulému provozu aplikace. Také obnova testovacích dat na stagingu neprobíhá pouze s nasazením nové verze, ale i nezávisle na ní. I tento proces může způsobit nefunkčnost prostředí. Jak jsme již zmínili, staging využívá i tým automatizovaného testování, který zde testuje framework pro psaní automatizovaných testů. Může se stát, že potřebují využít a otestovat komponentu určenou pro specifický region a v případě jeho nedostupnosti jsou tímto stavem blokováni.

Je tedy velmi přínosné mít přehledný stav dostupnosti e-shopů hlavních zemí, se kterými se denně pracuje a které zastupují svým nastavením jednotlivé regiony jako celek. Tímto minimalizujeme riziko zpomalení testovacího procesu a ve výsledku zpomalení celého procesu vývoje softwaru.

3.3 Existující nástroje řešící danou problematiku

Po celém světě existuje spousta jiných firem, které řeší podobný, ne-li stejný problém. Například weboví vývojáři mohou chtít být notifikováni v případě nedostupnosti vytvořených stránek, jelikož garantují klientům maximální plynulost jejich provozu. Z důvodu této poptávky existuje řada nástrojů, které danou problematiku řeší.

Každé řešení nabízené na trhu má svá specifika. Většina z nich jsou placená nebo nabízejí limitovanou službu zdarma a po překročení definovaných limitů je další užívání zpoplatněno. Výsledný přehled dostupnosti webových stránek bývá pevně daný, bez možnosti vlastní úpravy jak po vzhledové, tak po funkční stránce. Nyní se detailněji podíváme na několik aktuálně nabízených řešení.

3.3.1 UptimeRobot

Službu UptimeRobot nalezneme na stránkách https://uptimerobot.com/ a je nabízena za cenu limitací i zdarma. Každých pět minut nebo více posílá HTTP dotaz na server a o případné nedostupnosti umí notifikovat pomocí emailu, Slacku, telefonního hovoru nebo třeba zprávou v Microsoft Teams. Pokud budeme chtít využít bezplatnou verzi, máme možnost monitorovat maximálně padesát domén. Pro monitorování více domén a další funkcionality jako třeba kontrolu expirace SSL certifikátu či kratší intervaly monitorování je služba placená. UptimeRobot nabízí všem uživatelům velmi podrobnou dokumentaci, což je velké pozitivum.

3.3.2 Montastic

Montastic nalezneme na adrese https://montastic.com/ a opět máme možnost využít služby zdarma. Zde jsou však limitace větší a bez poplatků můžeme monitorovat pouze devět domén, a to každých třicet minut. To znamená, že stránka může být nedostupná plných dvacet devět minut, aniž bychom o tom byli informováni. Placená varianta nabízí možnost monitorovat až 500 domén, ale ne v kratších intervalech než pětiminutových, což při požadavku častých kontrol může být stále nedostačující. Montastic navíc nenabízí širší sortiment notifikací než skrze SMS zprávy nebo email. V čem však Montastic vyniká oproti většině nabízených řešení, je možnost vygenerování widgetů, které si uživatel může vložit na jakoukoli webovou stránku. Tento widget ve zjednodušené formě zobrazuje dostupnost našich webů.

3.3.3 Uptrends

Služba Uptrends nabízí monitoring internetových stránek, měření výkonnosti (například jak rychle se stránka načítá v různých prohlížečích), monitoring serverů a další. Je dostupná na adrese https://www.uptrends.com/. Velkou výhodou je nabídka digitálního hovoru se specialistou zdarma, který s klientem prodiskutuje jeho potřeby a ukáže mu možnosti služby. Nabídka notifikací je jedna z nejširších na trhu, a to pomocí emailu, SMS zprávy, telefonního hovoru, mobilní aplikace Slacku a spoustou dalších možností. Nevýhodou může být drahý měsíční poplatek, pokud budeme chtít využívat všech nabízených funkčností.

Uptrends nenabízí pouze monitoring dostupnosti webové stránky, ale dokáže měřit i její latenci, a to i pro různé webové prohlížeče. Při každém poslaném HTTP požadavku služba zachytí také rozsáhlá data o výkonnosti naší webové stránky.

V případě pomalého načítání nám dokáže vyhodnotit, jaká část stránky zpoždění způsobuje.

3.3.4 Pingdom

Jako poslední uvedeme službu Pingdom dostupnou na webových stránkách https://www.pingdom.com/. Tato služba je vyvíjena americkou firmou SolarWinds, která se mimo jiné zabývá právě monitoringem dostupnosti a výkonnosti IT infrastruktury. Využití služby zdarma je ale výrazně časově omezeno. Sofistikovaně je vymyšlen systém notifikací, který je odstupňován podle závažnosti nedostupností. Můžeme si nastavit určité domény jako velmi důležité a jejich nefunkčnost budeme notifikovat SMS zprávou nebo jako notifikaci do aplikace SolarWinds. Nastavit se dá taky možnost upozorňovat o nedostupnosti do té doby, než kompetentní osoba stvrdí, že je problém vyřešen.

Na trhu se mimo výše zmíněná řešení objevují i zcela neplacené produkty. Ty však mají zpravidla jiné nevýhody, a to například závislost na určité platformě, omezené funkcionality – notifikace pouze SMS zprávou, nutnost registrací a další, které zmíníme v následující kapitole spolu s vyhodnocením celé situace.

3.4 Zhodnocení situace

V předchozí kapitole jsme si ukázali, že trh s monitorujícími službami je poměrně nasycen. Přesto lze ale najít důvody, které ospravedlňují vývoj vlastní aplikace zabývající se monitoringem.

Začněme možnými nevýhodami spjatými s užíváním open-source (open-source software je software s přístupným zdrojovým kódem veřejnosti a některé licence dovolují, že do něj může kdokoli dobrovolně přispívat bez nároku na zisk) produktů. Bezplatné produkty založeny na filozofii open-source jsou vyvíjeny dobrovolníky, kteří mohou projekt opustit a přestat produkt udržovat. Paul Rubens (2014) ve svém článku uvádí, že zejména malé open-source projekty jsou spjaty s rizikem ztráty zájmu vedoucí osoby. Dále říká, že pokud tato situace nastane, nemusí být vůbec snadné najít jiného open-source vývojáře, který kormidlo převezme a v údržbě či rozvoji služby bude pokračovat. S touto skutečností okrajově souvisí i další nevýhoda bezplatného open-source softwaru. Khalil Khalaf (2017) tvrdí, že komunity uživatelů daného open-source softwaru samozřejmě existují, ale nemůžeme se na ně plně spoléhat, jelikož podpora, oprava chyb a tvorba instrukcí k používání softwaru není jejich zaměstnáním. Přes své

stinné stránky jsou bezplatné open-source produkty stále lákavé a pokud vše funguje, plní nám bez poplatků svou službu. V našem případě je tato potenciální nespolehlivost služby založené na filozofii open-source důvodem, proč tyto produkty nezvolíme jako řešení našeho problému.

Druhou existující variantou je placený software. Komerční služba odstraňuje nevýhody open-source softwaru, a tedy máme podporu od strany dodavatele a nemusíme se bát potenciálního ukončení fungování služby. Samozřejmě to není zadarmo a tato podpora a garance chodu softwaru přichází za úplatu. V předešlé kapitole jsme se blíže podívali na čtyři vybrané služby, které jsou založeny na přístupu SaaS (Software as a service, software jako služba). Gareth Goh (2015) ve svém článku uvádí, že zákazníci SaaS řešení typicky uzavírají roční nebo několikaleté smlouvy s dodavateli a poté hradí platby měsíčně. Placení výše zmíněných řešení tedy probíhá každý měsíc a uživatel řešení dostává jako službu po dobu, kdy je ochoten platit. Komerční software může být problémem zejména pro podniky ve chvíli finanční tísně.

Poslední variantou je vývoj vlastního softwaru. Negativem této varianty mohou být počáteční finanční a časové náklady potřebné k zrealizování aplikace. Ve chvíli, kdy je aplikace zprovozněna a splňuje očekávání, vynakládáme náklady potřebné pouze na její provoz (elektrická energie serveru). Ostatní náklady jsou nárazové, ať už jde o aktualizaci či o rozšíření služeb aplikace. Další výhodou je absolutní kontrola nad aplikací. Jelikož si ji sami vyvíjíme, můžeme její vzhled přizpůsobit podnikovým standardům a celkově řešení naimplementovat tak, abychom maximalizovali užitek pro konkrétní firmu. Další výhodou je bezpečnost. Webové stránky na testovacích prostředích nebo API mohou být zabezpečeny přihlašovacími údaji, které není v tomto případě nutné poskytnout třetí straně. V neposlední řadě nejsme závislí na cizím řešení a jakékoli úpravy nebo rozšíření můžeme komunikovat interně v podniku, což bývá většinou příjemnější a rychlejší než komunikace s technickou podporou externího řešení.

K řešení problematiky monitorování prostředí staging přistoupíme implementací vlastního řešení. Počáteční náklady na realizaci monitorující aplikace by neměly být natolik velké, aby bylo výhodnější soustavné měsíční placení poskytovateli externího řešení. Prostředí staging je dosažitelné kýmkoli na internetu, a proto je zabezpečeno nutností přihlášení. Při odesílání monitorujících HTTP požadavků za aktuálního nastavení prostředí je tedy potřeba odeslat i přístupové údaje spolu s požadavkem. Při

vlastním vývoji služby bude takto prováděno interně a zamezíme tak odhalení přihlašovacích údajů. Vlastní řešení je také více přizpůsobitelné a můžeme jej navrhnout i zrealizovat tak, aby přinášelo maximum užitku v kontextu našeho podniku a konkrétně námi řešené problematiky.

4 Návrh aplikace a její vývoj

Již máme základní teoretické základy a jsme obeznámeni s problematikou a řešením nabízeným na trhu. V této kapitole si popíšeme návrh a samotný vývoj aplikace.

4.1 Návrh

Ideálním scénářem každého projektu, a to nejen ve světě informačních technologií, je jeho úspěšné dokončení a splnění všech předem stanovených požadavků. Abychom maximalizovali šanci na úspěšné dokončení projektu, měli bychom před samotnou implementací zajistit požadavky, analyzovat je a vytvořit návrh či design budoucího řešení. Bett Obadia (2018) v časopise Project Management píše, že pokud má projekt špatně definovány cíle, je velmi pravděpodobné, že selže. Dalším aspektem je finanční hledisko v případě zjištění nesrovnalostí s požadovaným stavem od zákazníka. V čím ranější fázi projektu se tyto nesrovnalosti zjistí a odstraní, tím méně plýtváme zdroji. V této kapitole si shrneme požadavky na kýženou aplikaci a na jejich základě navrhneme optimální řešení.

4.1.1 Souhrn požadavků

Podnik požaduje automatizovaný monitoring domén, a to převážně těch, na kterých se provádí regresní testování a hrají důležitou roli v procesu vývoje. Nastavení časového intervalu, ve kterém bude probíhat monitoring (tzn. například kolikrát do hodiny je zkontrolována dostupnost), by měl být volitelný. Dále je žádoucí informovat v případě nefunkčnosti pomocí emailu tým, který může zahájit nápravu problematického stavu. Nastavení emailové funkcionality by mělo být konfigurovatelné, jelikož se může změnit odpovědný tým a tím i například cílová emailová adresa. Požadavek na načtení úvodní stránky e-shopu by měl navrátit status kód 200, v ojedinělých případech je akceptovatelný i status kód 301 (více o status kódech v kapitole 2.5.4.). Z toho vyplývá, že nedostupnost e-shopu je definována navrácením jiného status kódu serverem nežli 200 či 301. Pro přístup k e-shopům na testovacím prostředí je nutné uvést přihlašovací údaje, a to pomocí základní HTTP autentizace (Basic authentication). Podnik by měl mít také možnost jednotlivé domény spravovat, jelikož se množina důležitých e-shopů může v čase měnit. Je tedy žádoucí mít možnost domény do množiny přidávat či je z množiny odstraňovat. Posledním, ale neméně významným požadavkem je použití Microsoft technologií. Jelikož podnik

implementuje převážnou část řešení za využití .NET frameworku, programovacího jazyka C#, Microsoft SQL serveru a dalších, je žádoucí, aby byla daná aplikace vyvíjena v jazyce kompatibilním se zmíněnými technologiemi.

Požadavky na aplikaci jsou následující:

- Monitorovat automatizovaně dostupnost (definována jako navrácení status kódu 200 nebo 301) webových stránek v konfigurovatelném časovém intervalu.
- Monitorující služba musí mít možnost ověřit identitu, a to pomocí tzv. Basic HTTP authentication, jelikož pro interakci s doménami na testovacím prostředí staging je toto ověření vyžadováno.
- Zaslání emailové zprávy v případě nedostupnosti webových stránek. Cílová emailová schránka a samotná funkce by měly být konfigurovatelné.
- Funkce spravování domén, a to zejména mazání starých a přidávání nových.
- Jádro řešení musí být postaveno na Microsoft technologiích z důvodu snadné integrace do již existující platformy.

4.1.2 Analýza

Nyní, když máme v přehledné formě uceleny požadavky na výslednou aplikaci, je potřeba zanalyzovat celou situaci a navrhnout přístup, jak tyto požadavky uspokojit. Ve světě informačních technologií existuje zpravidla mnoho přístupů, jak řešit konkrétní problém. I v našem případě existuje řada možností.

Jedno z prvních rozhodnutí, které musíme učinit, je, zdali výsledné řešení bude aplikace desktopová nebo webová. Desktopová aplikace by běžela na klientovi, což v našem případě není ideální řešení. Monitorování domén by záviselo na skutečnosti, jestli má někdo desktopovou aplikaci spuštěnou. V případě, že by nikdo takový nebyl, monitoring nebude probíhat. Také chceme mít nad monitorováním kontrolu a být schopni běžící službu na serveru kdykoli vypnout a zapnout. Vypnutí dává smysl především před zahájením nasazování nové verze na staging, jelikož s velkou pravděpodobností budou domény nedostupné a výsledky monitoringu by neměly během nasazování verze vypovídající hodnotu. V našem případě je výhodnější řešení založit na webové aplikaci, která bude běžet na serveru. Na tomto serveru bude v konfigurovatelných časových intervalech spouštěn proces, který zkontroluje nadefinovaným doménám jejich dostupnost a výsledky uloží.

Funkce ukládání domén, které mají být monitorovány s informací o jejich aktuálních dostupnostech bude realizována s využitím databáze. Výhodou databáze je snadná manipulace s daty s využitím jazyka SQL. Pokud bychom data ukládali například v CSV souborech, nebylo by to v našem případě pohodlné, jelikož budeme data často číst, modifikovat, vytvářet a mazat. Za použití databáze si z ní server nejdříve vyčte seznam domén na zkontrolování, provede kontrolu a následně zaznamená informace o dostupnosti domén zpět do databáze.

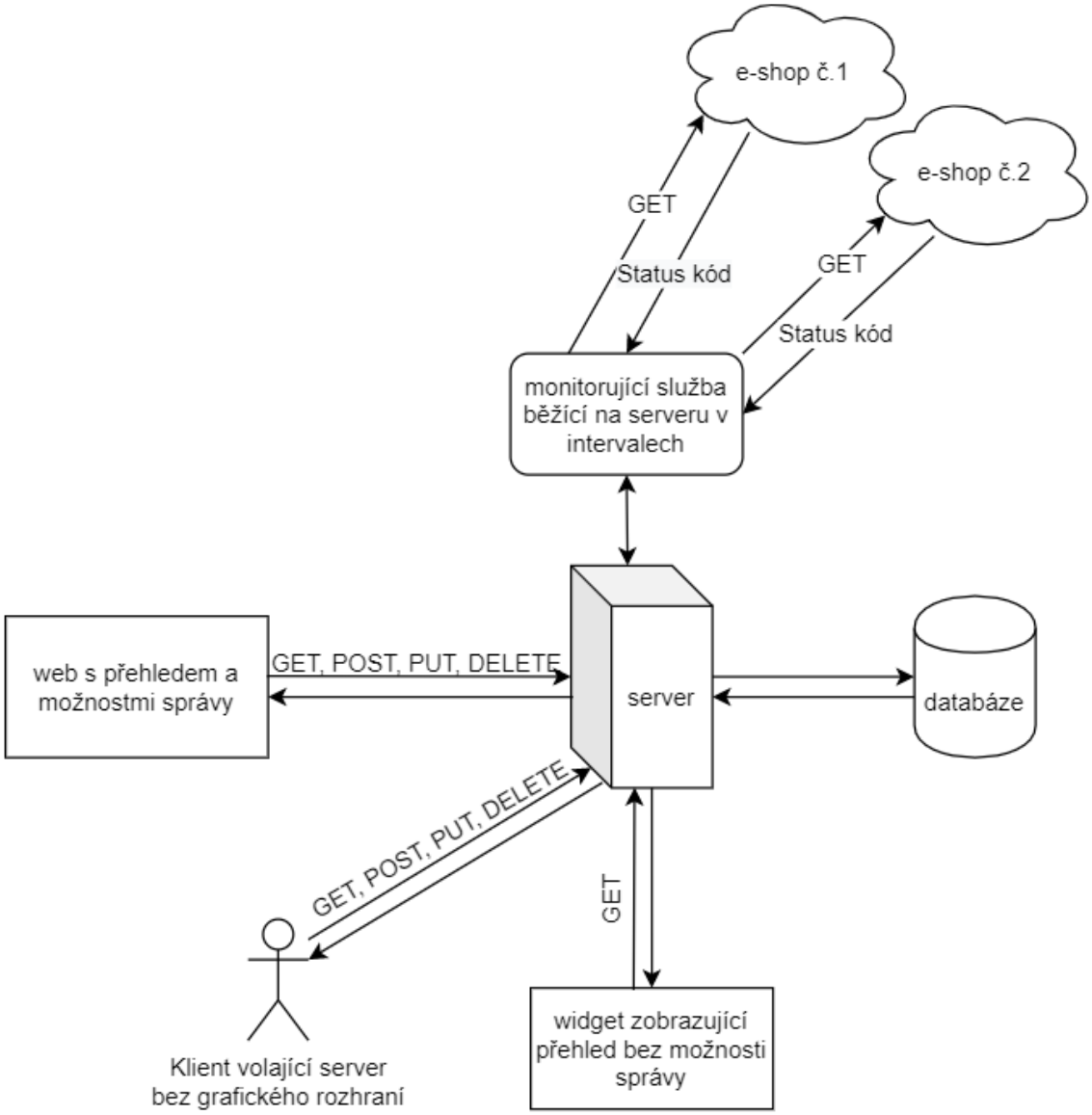
Zatím jsme si blíže nespecifikovali proces, jakým bude server získávat informace o dostupnosti. V požadavcích je specifikováno, že monitorované e-shopy musejí vracet status kód 200 nebo 301. Bude tedy potřeba, aby server posílal HTTP požadavky na dané e-shopy a kontroloval, zdali server e-shopu vrací požadovaný kód. V opačném případě zasílá emailovou zprávu o nedostupnosti e-shopu na zvolenou adresu pro zajištění nápravy.

Všechny konfigurace (emailová adresa, frekvence monitorování apod.) budou uloženy v souboru, který bude snadno přístupný a modifikovatelný. Jelikož budeme využívat databázi a pracovat s požadavky a odpověďmi protokolu HTTP, můžeme narazit na řadu chybových hlášení. Z tohoto důvodu je vhodné implementovat logovací funkcionalitu, kterou zajistíme vytvoření logovacího souboru obsahujícího informační, chybové a trasovací hlášení.

Nyní musíme promyslet, jakým způsobem bude moci uživatel upravovat seznam URL adres k monitorování. Jistě bude přínosné a komfortní mít webovou stránku, na které bude možnost spravovat URL adresy a zároveň mít přehled o jejich aktuálních dostupnostech. Přehled ve zjednodušené podobě (například bez možnosti správy) může být v budoucnu chtěný i na jiných stránkách, třeba formou widgetu. Měli bychom počítat i s takovými budoucími požadavky a udělat řešení natolik komplexní, abychom umožnili jednoduchou rozšiřitelnost.

Nejvýhodnějším přístupem pro vytvoření takto rozšiřitelné aplikace bude využití Web API, a tedy webové aplikace, která se chová jako služba a vyřizuje HTTP požadavky. Výhodné to bude z důvodu, že aplikace bude nabízet rozhraní, které může kdokoli zavolat a nechat se obsloužit naším serverem. To znamená, že kdokoli může požádat server o obdržení dat aktuálních dostupností, a server mu tuto informaci poskytne v textové podobě. Danou textovou podobu můžeme využít na několika

místech, třeba právě na webové stránce s přehledem, nebo jimi můžeme naplnit již zmiňovaný widget. Zkrátka si nad touto vrstvou vytvoříme grafické rozhraní podle naší libosti, které bude obdržené výsledky prezentovat v grafické podobě. Tato prezentační vrstva funguje nezávisle a v budoucnu může být naimplementována jinými technologiemi bez nutnosti změn na straně serveru. Jelikož slovní interpretace nemusí být vždy dostačující pro výslednou představu, celý navržený koncept můžeme vidět na obr. 4.1.



Obrázek 4.1 - Návrh architektury výsledného řešení (zdroj: vlastní)

Vidíme, že webovou aplikaci běžící na serveru kontaktuje nejen web zobrazující přehled a umožňující správu, ale také jej může kontaktovat klient bez grafického rozhraní. Motivací z klientovy strany může být například analyzování dostupností. Na obrázku dále vidíme i widget, který ve zjednodušené podobě zobrazuje data o

dostupnosti například v mobilní aplikaci, bez možnosti jejich správy. Z tohoto důvodu využívá pouze metody GET.

Jak vidíme, tento přístup nám přináší mnoho výhod. Služba je nezávislá na prezentační vrstvě a její klienti mohou s vyžádanými daty nakládat bez omezení.

4.1.3 Výběr použité technologie

Vzhledem ke stanoveným požadavkům musí být jádro aplikace založeno na technologiích společnosti Microsoft. Využijeme tedy Microsoft framework pro tvorbu webu ASP.NET, a to konkrétně ASP.NET Core. Prozatím použijeme databázi Microsoft SQL Server Express, jelikož je zdarma ke stažení i užívání. Tato verze obsahuje limitace, jako je například paměťový limit pro uložená data (maximum 10 GB), omezení automatizovaných záloh a tak dále. SQL Server Express je vhodný pro menší aplikace, pro které tyto limity nejsou svazující, což je i náš případ – naše databáze v případě patnácti uložených záznamů má velikost 16 MB.

Samotný web, který bude zobrazovat aktuální stav e-shopů a umožňovat přidávání, mazání či úpravu dat bude založen na HTML a CSS. Logika a propojení s API webové aplikace bude realizováno s využitím JavaScriptu, a to konkrétně s využitím Fetch API, která umožňuje velmi jednoduše a intuitivně posílat požadavky a zpracovávat odpovědi.

4.2 Backend

U většiny softwarů bychom mohli implementaci aplikace rozdělit na část backendovou a frontendovou. Backendovou částí chápeme část, která se nachází na straně serveru, zajišťuje jádro logiky aplikace, práci s daty a do jisté míry je uživateli skryta. Druhé části, kterou uživatel vidí a která představuje prezentační vrstvu na straně klienta, říkáme frontendová část. V této kapitole se podíváme na části našeho řešení, které řadíme do backendu.

4.2.1 Model a propojení s databází

Z našeho návrhu je zřejmé, že aplikace bude pracovat s databází. Zde budeme vkládat data o doménách či je modifikovat nebo odstraňovat. Zároveň budeme chtít data z databáze získávat, abychom mohli uživateli zobrazit jejich přehled. Pro tyto operace budeme využívat jazyka SQL, který je pro komunikaci s databází určen. Jednotlivé SQL dotazy nebudeme tvořit, protože pro práci s databází budeme používat Entity

Framework Core. Tento framework nám umožní pomocí objektu třídy, jež dědí z třídy DbContext (třída pocházející z Entity Framework Core), komunikovat s databází. Konkrétně budeme využívat takzvaný "code-first" přístup, což znamená, že si databázovou strukturu vytvoříme nejprve pomocí tříd v C# kódu. Vytvoříme si tedy model domény, který bude reprezentován třídou v jazyce C# a bude obsahovat všechny potřebné informace o doméně. Budeme potřebovat následující seznam informací.

- ID – Takzvaný primární klíč. V našem případě číslo, které bude v celém souboru domén unikátní právě pro jednu doménu a bude ji identifikovat. Tento údaj nesmí být prázdný, Entity Framework Core si jeho hodnotu automaticky vytváří samo.
- URL – Adresu, na kterou se bude posílat monitorující HTTP požadavek. V našem případě se bude jednat o adresu e-shopu. Tento údaj je povinný.
- Alias – Krátký, maximálně tři znaky dlouhý popisek dané domény. Alias budeme využívat na zaznamenání země, která je spojena s touto URL. Zemi budeme reprezentovat podle ISO normy, například pro mexický e-shop bude alias MX. V budoucnu se zde může značit například prostředí, na kterém se e-shop nachází, pokud tato informace nepůjde vyčíst již z URL (např. alias UAT pro prostředí UAT).
- Přístupové údaje (pokud jsou potřeba) – V databázi potřebujeme informaci o přihlašovacím jméně a hesle pro základní HTTP autentizaci. Hodnoty budeme uchovávat ve formátu Base64 jako textový řetězec, který se pak používá přímo v hlavičce požadavku klienta.
- Status kód – Esenciální informace, která naplňuje význam celé monitorující aplikace. Číselná hodnota navráceného status kódu serverem pro danou URL.
- Datum poslední aktualizace – Na závěr by bylo vhodné mít informaci o poslední provedené aktualizaci status kódu. V případě vyžádání dat z databáze nám hodnota data odhalí, jestli je status kód v kontextu času stále validním údajem.

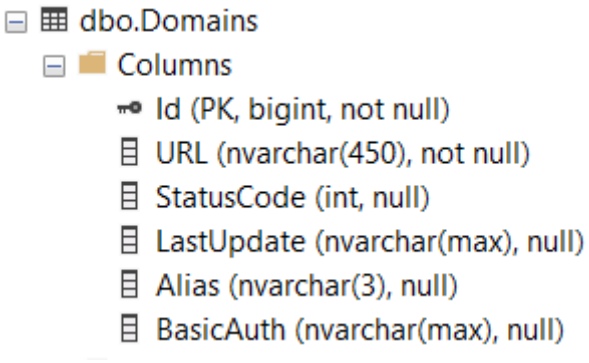
Na základě těchto informací vytvoříme třídu modelu domény. Výslednou třídu lze vidět na obr. 4.2.

```
/// <summary>
/// Model for representing domain
/// </summary>
12 references
public class Domain
{
    3 references
    public long Id { get; set; }
    [Required]
    5 references
    public string URL { get; set; }
    [MaxLength(3)]
    1 reference
    public string Alias { get; set; }
    10 references
    public string BasicAuth { get; set; }
    2 references
    public int? StatusCode { get; set; }
    2 references
    public string? LastUpdate { get; set; }
}
```

Obrázek 4.2 - Model domény, třída v C# (zdroj: vlastní)

Také vytvoříme třídu, jež je potomkem třídy DbContext. Nově vzniklou třídu pojmenujeme DomainContext a vytvoříme si v ní property datového typu generické třídy DbSet s parametrem našeho modelu Domain. Pomocí DomainContext budeme v aplikaci přistupovat k databázovým tabulkám. V této třídě také nadefinujeme unikátnost URL, abychom zamezili potenciálním duplicitám.

Nyní, když máme vytvořenou modelovou třídu reprezentující doménu a třídu dědící z DbContext, vytvoříme si novou migraci. Tímto krokem se vygeneruje SQL dotaz založený na kódu, který jsme naimplementovali. Migraci aplikujeme a tím dotaz vyhodnotíme. Tímto krokem se vše potřebné v databázi vytvoří. Máme tedy vytvořenou tabulku, do které se budou ukládat jednotlivé domény. Tato tabulka má sloupečky stejné, jako má modelová třída jednotlivé property. Tabulku v MS SQL databázi lze vidět na obr. 4.3.



Obrázek 4.3 - Databázová tabulka a její sloupečky (zdroj: vlastní)

Jelikož využíváme "code-first" přístup, kdykoli budeme potřebovat provést změny na straně databáze, musíme tak učinit změnou v kódu, vytvořením migrace a její následnou aplikací.

4.2.2 Monitorující služba

Třídu, která se zabývá monitorováním domén, můžeme nazvat jádrem celé aplikace. Právě zde, v předem nakonfigurovaných intervalech probíhá posílání požadavků na jednotlivé URL uložené v databázi a ukládání odpovědí zpět do databáze. Z tohoto místa se také volá metoda třídy, která obsluhuje zasílání emailů v případě nedostupnosti domén. Celá tato třída je implementována jako služba, která běží i bez nutnosti připojení klienta k serveru.

PingAndUpdateDB, což je název naší třídy, implementuje rozhraní IHostedService. Z tohoto důvodu musí implementovat metody StartAsync(CancellationToken) a StopAsync(CancellationToken). Jak už název napovídá, první zmiňovaná metoda by měla obsahovat kód, který spustí službu na pozadí. V našem případě si na začátku této metody zavoláme námi vytvořenou pomocnou metodu ConfigureMinutesAndEmail, která do předem připravených fieldů vloží uživatelovu konfiguraci emailu a frekvence monitoringu. Poté si vytvoříme instanci třídy Timer, pomocí níž budeme spouštět metodu TaskRoutine v intervalech. Implementace metody StartAsync můžeme spatřit na obr. 4.4.

```
0 references
public Task StartAsync(CancellationToken cancellationToken)
{
    ConfigureMinutesAndEmail();

    _timer = new Timer(TaskRoutine, null, 0, 60000 * _minutesToWait);
    return Task.CompletedTask;
}
```

Obrázek 4.4 - StartAsync metoda třídy, která implementuje rozhraní IHostedService (zdroj: vlastní)

V těle metody TaskRoutine se nachází logika samotného monitoringu. Pro každou doménu z tabulky si pomocí námi vytvořené statické třídy HttpPinger zavoláme HTTP dotaz a navrácený status kód od serveru si uložíme do proměnné httpStatusCode. Následně si ověříme, jestli navrácený status kód indikuje dostupnost domény či nikoli. Pokud doména není dostupná, přidáme ji do proměnných později využitých pro tvorbu notifikační emailové zprávy a zaznamenáme si skutečnost, že chceme zprávu odeslat. Bez ohledu na hodnotu status kódu vždy tento aktualizujeme v databázi spolu s datem o právě provedené aktualizaci. Logiku popsanou v tomto odstavci si můžeme prohlédnout na obr. 4.5.

```
foreach (Domain domain in listOfDomains)
{
    var httpStatusCode = (int)HttpPinger.GetHttpStatusCode(domain);

    // Save failed domain
    if (!_correctHttpResponseCodes.Contains(httpStatusCode))
    {
        _needSendEmail = true;
        _errorURLs.Add(domain.URL);
        _errorAliases.Add(domain.Alias);
        _errorCodes.Add(httpStatusCode);
    }

    // Modify domain and update in DB
    domain.StatusCode = httpStatusCode;
    domain.LastUpdate = DateTime.Now.ToString("HH:mm MM/dd");

    context.Entry(domain).State = EntityState.Modified;

    try
    {
        context.SaveChanges();
    }
    catch(DbUpdateConcurrencyException ex)
    {
        _logger.LogError("Record not existing anymore in db. Exception: " + ex);
    }
}
```

Obrázek 4.5 - Odesílání HTTP požadavku pro každou doménu a ukládní výsledků (zdroj: vlastní)

Popsanou třídu zaregistrujeme ve třídě Startup, a to konkrétně v metodě ConfigureServices.

4.2.3 Posílání emailů

Realizaci posílání notifikačních emailů zastřešují dvě třídy – EmailConfig a EmailSender. Třídu EmailConfig využijeme pro zredukování počtu parametrů při vytváření instance EmailSenderu. Pokud má konstruktor více než pět parametrů, bývá dobrým zvykem parametry zredukovat a předávat je v rámci jednoho objektu. EmailConfig obsahuje šest property, mezi které patří informace o SMTP serveru (adresa a číslo portu), přihlašovací jméno a heslo účtu, ze kterého se email odešle a taky emailová adresu odesílatele a příjemce. Validace těchto hodnot a vytvoření instance třídy EmailConfig se děje uvnitř metody ConfigureMinutesAndEmail, kterou jsme zmínili v kapitole 4.2.2. o monitorující službě.

Po úspěšném vytvoření instance EmailConfig třídy je tato ihned použita k vytvoření objektu třídy EmailSender. Tato třída obsahuje dvě metody. První z nich vytváří textovou zprávu emailu, která obsahuje informace o nedostupných URL, navrácených status kódech a čase provedení této kontroly. Obsah emailu vstupuje jako jeden z parametrů do druhé metody, která se stará o samotné odeslání emailové zprávy. K odeslání se využívá knihovny MailKit.

4.2.4 Controller a vytvoření API

Za účelem získávání a modifikace dat v databázi ze strany klienta, jak jsme znázornili na obrázku 4.1, je potřeba vytvořit kontroler. V našem případě se jedná o třídu DomainsController, která dědí z třídy ControllerBase. Zde nadefinujeme asynchronní metody, které budou zavolány reakcí na požadavek klienta. Metoda se zavolá podle HTTP metody požadavku, cílové adresy a volitelných parametrů. Kupříkladu HTTP požadavek s metodou DELETE cílený na relativní adresu /api/v1/Domains/{id}, kde id je id domény potřebné ke smazání zavolá metodu zobrazenou na obr. 4.6.

```
[HttpDelete("{id}")]
0 references
public async Task<ActionResult<Domain>> DeleteDomain(long id)
{
    var domain = await _context.Domains.FindAsync(id);
    if (domain == null)
    {
        return NotFound();
    }

    _context.Domains.Remove(domain);
    await _context.SaveChangesAsync();

    return domain;
}
```

Obrázek 4.6 – Ukázka metody třídy DomainsController. Metoda obsluhuje DELETE http požadavek (zdroj: vlastní)

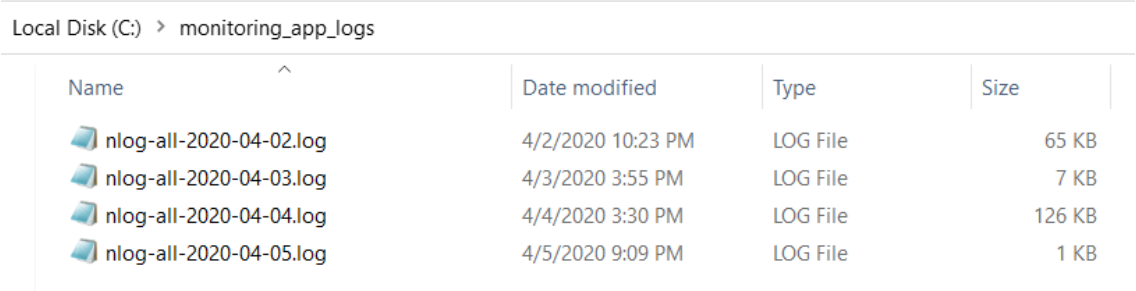
4.2.5 Konfigurace aplikace a logování

Aby si uživatel mohl nastavit frekvenci monitorování nebo vypnout či zapnout emailové notifikace, jsou tyto funkce v aplikaci parametrizovány. Hodnoty parametrů vstupují do jednotlivých funkcí ze souboru appsettings.json, ve kterém je konfigurace obsažena. Nastavení je zapsáno ve formě páru klíč - hodnota. V souboru appsettings.json je také konfigurace pro spojení s databází. Nastavení monitorující frekvence a zasílání e-mailů je znázorněna na obr. 4.7.

```
//Configuration for the service
"minutesToWait": 1, // amount of minutes to wait between http requests, by default 1 min
"sendNotificationEmail": true, // set true if you need to send email when some domain is down
"SMTPserverAddress": "smtp.gmail.com",
"SMTPserverPort": 587,
"userLoginName": "loginName",
"userPassword": "somePassword",
"emailFrom": "emailFrom@gmail.com",
"emailTo": "InfraOpsTeam@gmail.com"
```

Obrázek 4.7 – Přehled nastavitelné konfigurace pro frekvenci monitorování a zasílání e-mailových zpráv (zdroj: vlastní)

Pro logování do textového souboru využijeme balíčku NLog.Web.AspNetCore. Logování využíváme především k zaznamenávání hlášek při chybné validaci konfigurací nebo při chybových stavech aplikace. Nastavení logování provedeme v souboru nlog.config, kde jsme mimo jiné nastavili cílovou lokaci logovacího souboru na C:\monitoring_app_logs. Názorné vytvoření logovacích souborů je na obr. 4.8.



Obrázek 4.8 - Vytvořené logovací soubory (zdroj: vlastní)

4.3 Frontend

V této podkapitole se zaměříme na frontend naší aplikace. Ukážeme si její vzhled, jak prezentační vrstvu tvoříme a jakým způsobem komunikuje s backendovou částí aplikace. Na závěr se podíváme na responzivitu celého designu.

4.3.1 Designové prvky aplikace

Vzhled aplikace spuštěné v prohlížeči na desktopu můžeme zhlédnout na obr. 4.9.

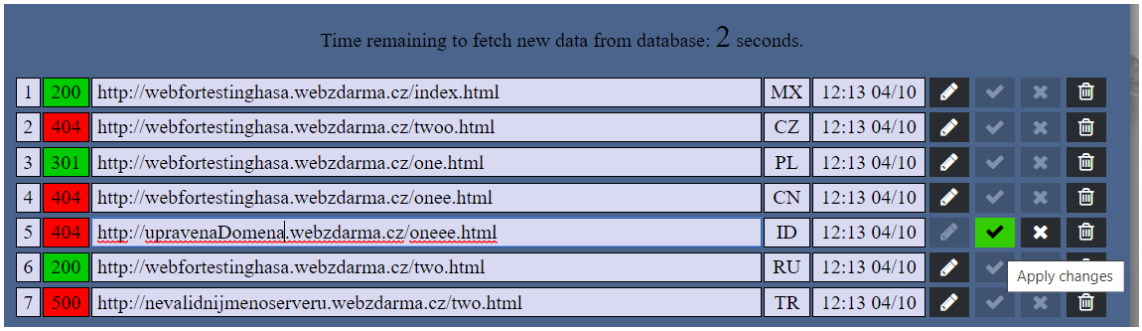
Obrázek 4.9 - Vizuální stránka aplikace spuštěné na desktopu (zdroj: vlastní)

Aplikaci můžeme na pomyslné vertikální ose rozdělit do čtyř částí. V první části pod názvem máme tři formulářové inputy. Do prvního vkládáme URL, druhý input slouží pro alias a do třetího vkládáme přihlašovací údaje pro základní HTTP autentizaci. Pouze první pole je povinné a bez jeho vyplnění nám aplikace neumožní doménu přidat. Současně se při odeslání kontroluje formát URL a v případě nevalidní URL adresy není adresa do seznamu přidána a uživatel je o této skutečnosti notifikován. Za formulářovými vstupy máme ikonu oka, která po kliknutí zobrazí hodnotu třetího

inputu, jež je za normálních okolností skryta za hvězdičkami. Tlačítko se symbolem plus formulář stvrzuje a odesílá na stranu serveru.

Ve druhé části z vrchu se nachází odpočet, který uživateli zobrazuje, za jak dlouho se stáhnou nová data z databáze k zobrazení. Data se stahují každých třicet vteřin.

Třetí částí je samotné zobrazení domén. Nalevo máme pořadí, navrácený status kód s barevným podbarvením a URL domény. Vpravo je pak tříznakový popisek, čas s datem kontroly a čtyři ikony sloužící pro editaci. Po kliknutí na ikonu pera máme možnost upravit pole URL a aliasu. Na obr. 4.10 jsme páté doméně upravili hodnotu URL a chystáme se úpravu potvrdit.



Obrázek 4.10 - Potvrzení úpravy editace URL na desktopu (zdroj: vlastní)

Poslední částí je tlačítko, po jehož kliknutí se nestahují nová data z databáze, ale samotný klient všechny zobrazené domény zkontroluje a výsledky poté neaktualizuje v databázi, ale v prezentační vrstvě. Tlačítko může využít uživatel, který nechce čekat na obdržená data z databáze nebo si chce okamžitě ověřit, zdali doména nebo skupina domén je stále nedostupná a informace z databáze je aktuální.

4.3.2 Dynamické vytváření obsahu

V této podkapitole se podíváme, jak výše popsaný design vytváříme. Prezentační vrstva je tvořena pomocí třech souborů.

Prvním souborem je HTML, v jehož hlavičce máme reference na soubory třetích stran. Prvním z nich je CSS soubor Font Awesome, který do aplikace dodává ikony tlačítek. Dalším souborem v hlavičce je JavaScript knihovna, kterou využíváme na validaci URL adresy na straně klienta. V těle souboru HTML je nadefinován formulář s inputy, odpočet času zbývajícího do stažení dat z databáze, tlačítko pro okamžitý monitoring a prázdný `<div>` element. Do tohoto prázdného elementu budeme

dynamicky tvořit pomocí JavaScriptu jednotlivé řádky a sloupce s informacemi o doménách.

Druhý soubor obsahuje kód JavaScriptu. Zde zastřešujeme logiku na straně klienta pomocí funkcí, které se volají po kliknutí na tlačítka nebo v časových intervalech (např. odpočet). Při načtení stránky pomocí `window.onload` čekáme, až se na pozadí stáhnout data z databáze a stránka se dynamicky vygeneruje. Ve chvíli načtení stránky zavoláme funkce na synchronizaci časů odpočtu a intervalu stahování dat z databáze. Poté se po uplynutí každých třiceti vteřin volá funkce `getDomains()`, která pomocí fetch API zavolá GET požadavek a odpověď v JSON formátu dále předá funkci `showDomains(domains)`. Funkce `showDomains(domains)` s využitím DOM přistupuje k HTML elementům a dynamicky vytváří obsah stránky do předem připraveného prázdného `<div>` elementu. Funkce iteruje skrze obdržené domény a pro každou z nich vytvoří postupně celý řádek a všechny sloupce (pořadí, URL, …). Během generování si ke každému tlačítku a divu, který chceme později umožnit editovat přidáme atribut `data-domainid` a jeho hodnotu nastavíme na id domény, ke které tlačítka patří. Pokud uživatel klikne na jakékoli tlačítko pro správu domén umíme si pomocí tohoto atributu zjistit o jakou doménu se jedná a odemknout správné elementy pro editaci.

Třetím souborem vytvářejícím prezentační vrstvu je soubor CSS, který obsahuje stylování celé stránky a také definuje responzivitu designu, na kterou se zaměříme v následující podkapitole.

4.3.3 Responzivita designu

V roce 2020 jsou mobilní zařízení a tablety spojeny s každodenním životem většiny populace vyspělých zemí. Nepřímo to dokazuje Eric Enge (2019) v článku čerpající z dat společnosti SimilarWeb, která mimo jiné analyzuje provoz na internetu. Eric Enge (2019) v článku znázorňuje, že pro monitorovanou množinu webových stránek byla návštěvnost téměř 60 % z mobilních zařízení. Tento trend je zdá se rostoucí v čase a z tohoto důvodu bychom měli i my přizpůsobit design mobilním zařízením.

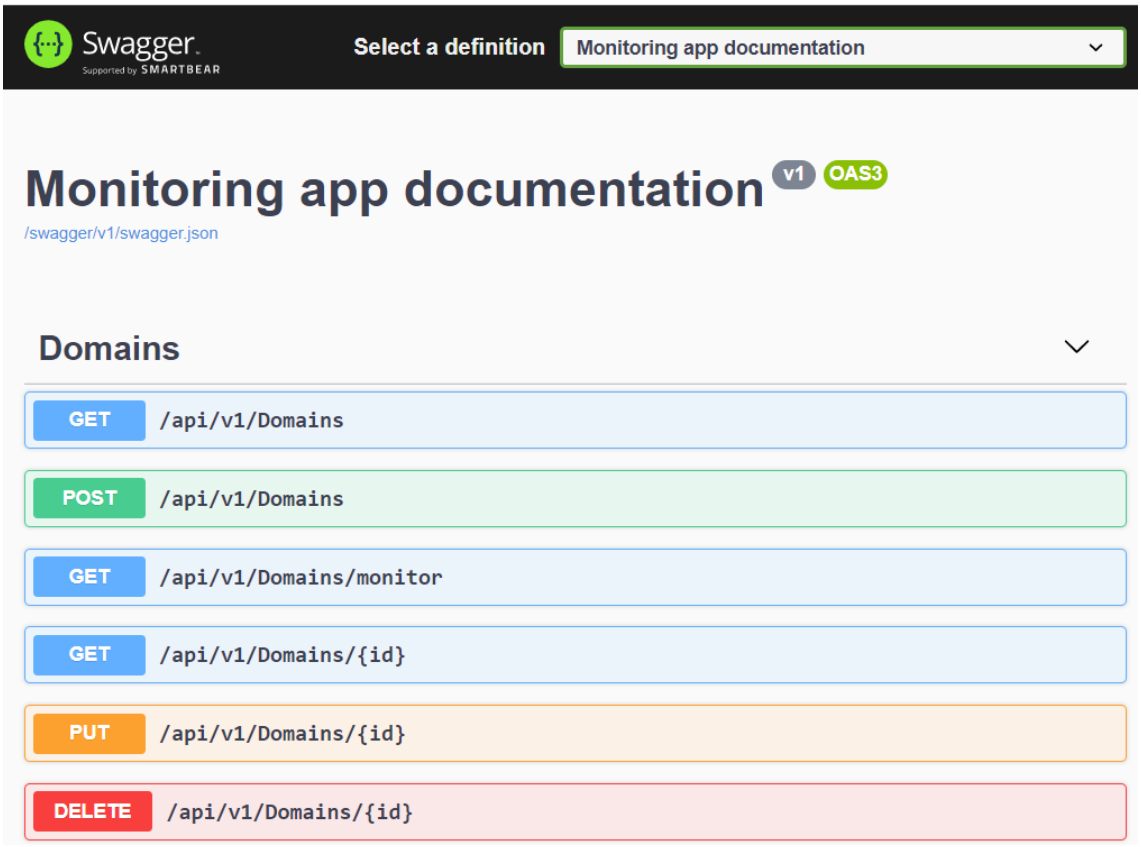
Adaptaci vzhledu stránky na měnící se rozlišení naimplementujeme pomocí atributu media. Tento atribut nám dovoluje specifikovat styly pro různá rozlišení. Styly jsou pak aplikovány s prioritou, pokud se aplikace nachází v definovaném rozlišení. Na obr. 4.11 je znázorněna naše aplikace v rozlišení mobilního telefonu iPhone X. V levé

části obrázku je stránka po načtení a napravo je monitorující část po přejetí prstem směrem doleva.

Obrázek 4.11 - Nalevo i napravo aplikace zobrazená na zařízení iPhone X (zdroj: vlastní)

4.4 Dokumentace

Pro zjednodušení komunikace se serverem jsme vytvořili za použití populárního nuget balíčku Swashbuckle.AspNetCore dokumentaci naší API. Implementace tohoto řešení je velmi jednoduchá a dokumentace se generuje automaticky na základě kontrolerů a modelů. Pro vytváření je potřeba přidat službu v třídě Startup a ve stejné třídě přidat middleware komponentu. Dokumentace je dostupná po přidání relativní cesty /swagger/index.html k adrese služby. Všechny dostupné endpointy zobrazuje v přehledném schématu, které je ke zhlédnutí na obr. 4.12.



Obrázek 4.12 - Dokumentace endpointů API (zdroj: vlastní)

Po rozkliknutí jednotlivých endpointů je možné těchto ihned využít a po odeslání požadavku se zobrazí i odpověď serveru. Mimo dokumentaci se tedy jedná i o jednu z možností získávání a modifikování domén.

Spolu s tímto přehledem máme v dokumentaci i informaci o modelu domény. Díky tomu víme, v jakém formátu můžeme odpověď očekávat i v jakém formátu posílat tělo požadavku v případě modifikace domény. Povinné property jsou označeny hvězdičkou, v našem případě je povinné pouze uvedení URL. Zobrazené schéma modelu v dokumentaci si můžeme prohlédnout na obr. 4.13.

Obrázek 4.13 - Schéma modelu domény vygenerované v dokumentaci (zdroj: vlastní)

5 Závěr

Cílem práce bylo vyvinutí monitorující aplikace, která bude poskytovat přehled dostupnosti webových stránek a notifikovat formou emailové zprávy v případě jejich nedostupnosti. Definovaný cíl byl splněn. Ve druhé kapitole jsme si uvedli teoretické informace o technologiích a přístupech, které byly v praktické části použity. Díky třetí kapitole jsme získali povědomí o aktuální problematice a zmapovali jsme nabídky dostupných řešení na trhu. V praktické části jsme nejprve provedli analýzu nashromážděných požadavků na aplikaci a vytvořili návrh řešení. Tento návrh jsme v dalších kapitolách převedli do reality, a to za použití jazyka C# a technologií společnosti Microsoft na straně serveru. Konkrétně řešení využívá Microsoft SQL Express databázi, ASP.NET Web API a Entity Framework Core. S využitím frameworku ASP.NET Web API jsme vytvořili HTTP službu, kterou napojením na prezentační vrstvu vytváříme webovou stránku s přehledem a možností správy internetových stránek určených k monitorování. Díky využití technologie ASP.NET Web API a vytvořením HTTP služby nabízíme komukoli možnost si data vyžádat nebo modifikovat s využitím jiné prezentační vrstvy dokonce i s její absencí. Naše webová stránka s přehledem a správou využívá HTML a CSS. Komunikace se serverem a dynamické vytváření jejího obsahu je implementováno s využitím JavaScriptu.

Aktuálně aplikace během monitorování umožňuje pouze provádění základního HTTP ověření, což plně dostačuje momentálním požadavkům. V budoucnu však může nastat situace, kdy cílová monitorující webová stránka bude zabezpečena jiným ověřením. V takovém případě je potřeba aplikaci rozšířit a umožnit i ověřování tímto způsobem. Případné změny by znamenaly rozšíření databázové tabulky, která by musela obsahovat informaci o typu zabezpečení pro danou stránku. Zároveň by v databázi musel být uložen ověřovací token či jinak zahashované přihlašovací údaje. V neposlední řadě by změna zasahovala do třídy, která posílá monitorující požadavky. Tato by musela umožňovat přidávání nových typů ověřování do HTTP požadavku.

Také není nijak ošetřeno přidělování práv pro správu webových stránek určených k monitorování. Aktuálně může kdokoliv s přístupem do podnikové sítě pomocí našeho webu nebo přímo komunikací s HTTP službou nejen číst, ale i modifikovat, mazat a vytvářet nové záznamy. Služba bude spuštěna na serveru v lokální síti, tedy nebude dosažitelná z veřejného internetu. Přesto však může být přínosné

kontrolovat správu domén a nepovolit tyto možnosti každému, kdo má přístup do lokální sítě podniku. Z tohoto důvodu jako potencionální zlepšení vidím naimplementování funkcionality přihlašování a umožnění operací, které jakkoli mění hodnoty v databázi až po uživatelově přihlášení. Dané rozšíření se dá zrealizovat přidáním Identity serveru a zasíláním JWT tokenu v hlavičce požadavků.

Seznam použité literatury

Odborná kniha

PILGRIM, Mark. *Ponořme se do HTML5.* Praha: CZ.NIC, z. s. p. o., 2014. ISBN 978-80-905802-6-8.

GOODYEAR, Steve. *Practical SharePoint 2013 Governance.* Berkeley: Apress, 2013. ISBN 978-1-4302-4887-3.

DESIKAN, Srinivasan a Ramesh GOPALASWAMY. *Software Testing: Principles and Practice.* Delhi: Dorling Kindersley, 2006. ISBN 978-8177581218.

DUCKETT, Jon. *HTML and CSS: Design and Build Websites.* Indianapolis: John Wiley&Sons, Inc., 2011. ISBN 978-1-118-00818-8.

MARINI, Joe. *Document Object Model.* United States: The McGraw-Hill Companies, Inc. 2002. ISBN 978-0-07-222831-1.

CORNER, E. Douglas. *Computer Networks and Internets.* 5. vyd. New Jersey: Pearson, 2009. ISBN 978-0136061274.

RODRIGUEZ, A., J. GATRELL, J. KARAS and R. PESCHKE. *TCP/IP Tutorial and Technical Overview (IBM Redbooks).* 7. vyd. New York: IBM Corporation, 2001. ISBN 9780738421650.

LUDIN, Stephen and Javier GARZA. *Learning HTTP/2: A Practical Guide for Beginners.* Sebastopol: O'Reilly Media Inc., 2017. ISBN 978-1491962442.

MARRS, Tom. *JSON at Work.* Sebastopol: O'Reilly Media Inc., 2017. ISBN 978-1-449-35832-7.

RISCHPATER, Ray. *JavaScript JSON Cookbook.* Birmingham: Packt Publishing Ltd., 2015. ISBN 978-1-78528-690-2

CLARK, Dan. *Beginning C# Object-Oriented Programming (2nd Edition).* New York: Apress, 2013. ISBN 978-1430249351.

KRIEGEL, Alex. *Discovering SQL: A Hands-On Guide for Beginners.* Hoboken: Wiley Publishing, Inc., 2011. ISBN 978-1-118-00267-4.

Článek v odborném časopise nebo ve sborníku z konference

OBADIA, Bett. Why Projects Fail. *Project Management Journal*. 2018, č. 1, s. 7. ISSN 8756-9728.

Elektronické dokumenty a ostatní

RUBENS, Paul. CIO: *7 Reasons Not to Use Open Source Software* [online]. CIO [11. 02. 2014]. Dostupné z: https://www.cio.com/article/2378859/7-reasons-not-to-use-open-source-software.html

KHALIL, Khalaf. MEDIUM: *The Pros and Cons of Open Source Software* [online]. MEDIUM [11.06.2017]. Dostupné z: https://medium.com/4thought-studios/the-pros-and-cons-of-open-source-software-d498304f2a95

GOH, Gareth. INSIGHTSQUARED: *SaaS Billing: Why You Want Upfront Payments* [online]. INSIGHTSQUARED [2015]. Dostupné z: https://www.insightsquared.com/blog/saas-billing-why-you-want-upfront-payments/

ELLIOTT, Eric. MEDIUM: *How Popular is JavaScript in 2019?* [online]. MEDIUM [11.05.2019]. Dostupné z: https://medium.com/javascript-scene/how-popular-is-javascript-in-2019-823712f7c4b1

GAMAGE, Ashen Thilina. MEDIUM: *Evolution of HTTP — HTTP/0.9, HTTP/1.0, HTTP/1.1, Keep-Alive, Upgrade, and HTTPS* [online]. MEDIUM [17.11.2017]. Dostupné z: https://medium.com/platform-engineer/evolution-of-http-69cfe6531ba0

ENGE, Eric. PERFICIENT DIGITAL: *Where is the Mobile vs. Desktop Story Going?* [online]. PERFICIENT DIGITAL [11.04.2019]. Dostupné z: https://www.perficientdigital.com/insights/our-research/mobile-vs-desktop-usage-study

Seznam zkratek

HTML – Hypertext Markup Language

CSS – Cascading Style Sheets

DOM – Document Object Model

CMS – Content Management System

API – Application Programming Interface

SaaS – Software As A Service

DTAP – Development, Testing, Acceptance and Production

OSI – Open Systems Interconnection

ISO – International Organization for Standardization

URL – Uniform Resource Identifier

MIME – Multipurpose Internet Mail Extensions

JSON – JavsScript Object Notation

CSV – Comma Separeted Values

SQL – Structured Query Language

GB – Gygabyte

REST – Representational State Transfer

EF Core – Entity Framework Core

Prohlašuji, že

- jsem byl(a) seznámen(a) s tím, že na mou diplomovou (bakalářskou) práci se plně vztahuje zákon č. 121/2000 Sb. – autorský zákon, zejména § 35 – užití díla v rámci občanských a náboženských obřadů, v rámci školních představení a užití díla školního a § 60 – školní dílo;
- beru na vědomí, že odevzdáním diplomové (bakalářské) práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že Vysoká škola báňská – Technická univerzita Ostrava (dále jen VŠB-TUO) má právo nevýdělečně, ke své vnitřní potřebě, diplomovou (bakalářskou) práci užít (§ 35 odst. 3);
- souhlasím s tím, že diplomová (bakalářská) práce bude v elektronické podobě archivována v Ústřední knihovně VŠB-TUO. Souhlasím s tím, že bibliografické údaje o diplomové (bakalářské) práci budou zveřejněny v informačním systému VŠB-TUO;
- bylo sjednáno, že s VŠB-TUO, v případě zájmu z její strany, uzavřu licenční smlouvu s oprávněním užít dílo v rozsahu § 12 odst. 4 autorského zákona;
- bylo sjednáno, že užít své dílo, diplomovou (bakalářskou) práci, nebo poskytnout licenci k jejímu využití mohu jen se souhlasem VŠB-TUO, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly VŠB-TUO na vytvoření díla vynaloženy (až do jejich skutečné výše).

V Ostravě dne 14.5.2020

Matěj Haša

Seznam příloh

Příloha 1 : archiv se spustitelnou aplikací.

Příloha č. 1

Příloha obsahuje:

- zdrojový kód aplikace